Python WSRF Programmers' Tutorial

Joshua Boverhof

Python WSRF Programmers' Tutorial

Joshua Boverhof
Copyright © 2005 Joshua Boverhof

This tutorial is available for use and redistribution under the terms of the Globus Toolkit Public License [http://www-unix.globus.org/toolkit/license.html]

This tutorial is an adaptation of Borja's GT4 Tutorial for the pyGridWare framework, most of the contents were directly copied.

Table of Contents

List of Figures

Introduction

This document is intended as a starting point for anyone who is going to develop applications using pyGridWare.

The tutorial is divided into 2 main areas:

- **Getting Started**: An introduction to key concepts related with GT4 and the Web Services Resource Framework (WSRF)
- **GT4 Python Core**: A guide to programming basic Web Services which only use the Python WS Core component of GT4.

Please note that the tutorial, at this point, only covers the *Python WS Core* component of the toolkit, an important but *small* part of the whole toolkit. At the end of this tutorial you will know how to program stateful Web services using GT4. This will allow you to progress towards using the higher-level services of the toolkit (using official Globus documentation). However it is important to understand that *you cannot program Grid-based applications using* only *the Python WS Core component of the toolkit*. This tutorial should be approached as a stepping stone towards more powerful tooling, not as a definite guide on Grid programming.

GT4 Prerequisite Documents

This tutorial has no GT4 prerequisite documents, since it is intended as a starting point for GT4 programmers. However, you should already be familiar with Grid Computing. A good, short introduction to what Grid Computing is can be found in Ian Foster's paper The Grid: A New Infrastructure for 21st Century Science [http://www.aip.org/pt/vol-55/iss-2/p42.html]. A more extense, and very easy to read, introduction can be found at the Grid Café [http://gridcafe.org/].

For a much more detailed text, you might want to check out the following book: The Grid 2: Blueprint for a New Computing Infrastructure [http://www.mkp.com/grid2] (Edited by Ian Foster and Carl Kesselman. 2003). Most of the book is easy to read and not too technical. It is also known as "The Grid Bible". With a name like that, you can assume it's worth taking a look at it :-)

You might also be interested in taking a look at the 'Publications' section in the Globus website [http://www.globus.org], specially the documents listed below. However, these documents are rather technical and might be too hard for a beginner. You might want to just skim through them at first, and then reread them once you're familiar with GT4.

- The Anatomy of the Grid: Enabling Scalable Virtual Organizations [http://www.globus.org/research/papers/anatomy.pdf] . I. Foster, C. Kesselman, S. Tuecke.
- The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration [http://www.globus.org/research/papers/ogsa.pdf] . I. Foster, C. Kesselman, J. Nick, S. Tuecke.

Audience

This document is intended for programmers who are new to the Globus Toolkit 4 (GT4).

Assumptions

The following knowledge is assumed:

- Programming in Python. If you don't know Python, you can find some useful links here.
- Basic knowledge of XML. If you have no idea of XML, you can find some useful links here.
- You should know your way around a UNIX system. This tutorial is mainly UNIX-oriented, although in the future we hope to include sections for Windows users.
- Basic knowledge of what the Grid and grid-based applications are. This tutorial is not intended as an introduction to Grid Computing, but rather as an introduction to a toolkit which can enable you to program grid-based applications.

The following knowledge is *not* required:

- Web Services. The tutorial includes an introduction to fundamental Web Services concepts needed to use GT4.
- Globus Toolkit 2 or 3

Related Documents

- Specifications
  - OASIS WSRF Page [http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf]: The most recent version of the WSRF specifications can be found here.

    Globus WSRF Page [http://www.globus.org/wsrf/]: Contains pointers to papers and presentations related to WSRF.
- Official Globus Documentation
  - GT4 Fact Sheet [http://www-unix.globus.org/toolkit/GT4Facts/]: Everything there is to know about GT4.
  - Official GT4.0 Documentation [http://www.globus.org/toolkit/docs/4.0/]: Includes the installation guide.
- Other related documents
  - Ian Foster's Globus Toolkit Primer [http://www.globus.org/toolkit/docs/4.0/key/GT4_Primer_0.6.pdf] provides a detailed look at the toolkit and all the components included in it.

Document Conventions

The following conventions will be observed throughout this document:

Code

```
class HelloWorld:
  def __call__(self, *args, **kw):
     ❶
     # Code in bold is important
     print "Hello World"
```

```
HelloWorld()()
```

(1) This is a callout, further explaining an important part of the code.

Inlined code

Whenever we refer to bits of code from the main text, it will be highlighted `like this`. For example:

> The `HelloWorld` class has a `__call__` method that prints out a `"Hello World"` string.

Shell commands

```
python HelloWorld.py
```

If a command is too long to fit in a single line, it will be wrapped into several lines using the backslash ("\") character. On most UNIX shells (including BASH) you should be able to copy and paste all the lines at once into your console.

```
python \
HelloWorld.py \
arg1 \
arg2
```

Notes

You can find three types of notes in the text: complementary information, reminders, and warnings.

Tip

This is a complementary information block.

This kind of note contains interesting information that complements what is currently being discussed in the text.

Note

This is a reminder.

This kind of notes are usually used after a block of code to remind you of where you can find the file that contains that particular code. It is also used to remind you of important concepts, and to suggest what sections of the tutorial you should read again if you have a hard time understanding a particular section.

Caution

This is a warning.

Warnings are used to emphatically point out something. They generally refer to common pitfalls or to things that you should take into account when writing your own code.

About the author & acknowledgments

The Python WSRF Programmer's Tutorial was adapted by Joshua Boverhof [http://dsd.lbl.gov/~boverhof/] from Borja Sotomayor's [http://people.cs.uchicago.edu/~borja/] GT4 Toolkit Java WS-Core Programmer's Tutorial.

Acknowledgments

The following people have, in one way or another, helped with the GT4 Tutorial:

- Borja Sotomayor's [http://people.cs.uchicago.edu/~borja/]
- Lisa Childers
- Rebeca Cortazar [http://paginaspersonales.deusto.es/cortazar/]
- Ian Foster [http://www-fp.mcs.anl.gov/~foster/]
- Leon Kuntz (in memoriam)
- Jesus Marco
- All the Globus gurus who have reviewed the tutorial on countless occasions

readers help to improve the tutorial by reporting bugs and typos, as well as making very constructive comments and suggestions:

If you've reported a bug, typo, or helped out in any way, and you are not listed here, please do let me know!

Part I. Getting Started

Table of Contents

Chapter 1. Key Concepts

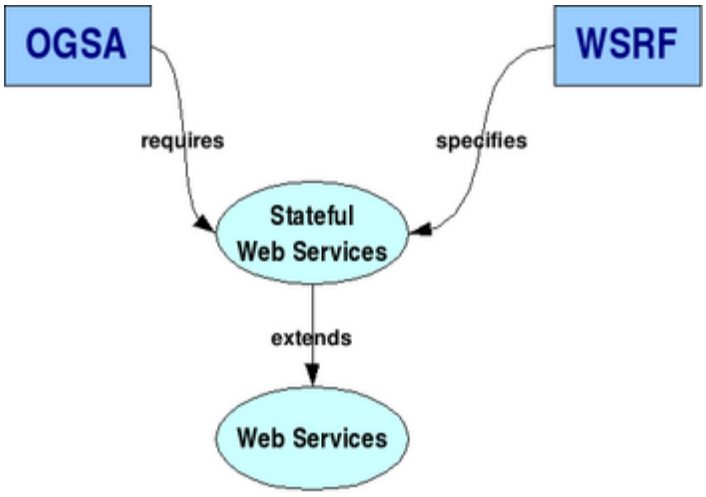
There are certain key concepts that must be well understood before being able to program with GT4. This chapter gives a brief overview of all those fundamental concepts.

- **OGSA, WSRF, and GT4**: We'll take a look at what these oft-mentioned acronyms mean, and how they are related.
- **Web Services**: OGSA, WSRF, and GT4 are based on standard Web Services technologies such as SOAP and WSDL. You don't need to be a Web Services expert to program with GT4, but you should be familiar with the Web Services architecture and languages. We provide a basic introduction and give you pointers to interesting sites about Web Services.
- **The Web Services Resource Framework**: WSRF is the core of GT4. We take a look at what a Web Service Resource (or WS-Resource) is, and how it is related to Web Services.
- **The GT4 Architecture**: After seeing both WS-Resources and Web Services, we take a look at the whole GT4 architecture, and how WSRF fits in it.
- **Java & XML**: Finally, if you want to use GT4, you need to be able to program in Java, and to understand basic XML. If you're new to Java and XML, we provide a couple links that can help you get started.

OGSA, WSRF, and GT4

If you've started looking at GT4, you've probably encountered at least these two acronyms: OGSA and WSRF. But... what do they mean? How are they related to the Globus Toolkit 4? This section attempts to clarify these important concepts. First of all, let's start by taking a look at OGSA and WSRF *without* seeing just yet how they are related to GT4.

Figure 1.1. Relationship between OGSA, WSRF, and Web Services



OGSA

A grid application will usually consist of several different components. For example, a typical grid application could have:

- **VO Management Service**: To manage what nodes and users are part of each Virtual Organization.

- **Resource Discovery and Management Service:** So applications on the grid can discover resources that suit their needs, and then manage them.
- **Job Management Service**: So users can submit tasks (in the form of "jobs") to the Grid.
- And a whole other bunch of services like security, data management, etc.

Note

If you have absolutely no idea of what I have just said (and feel slightly confused), remember that the tutorial assumes that you know what Grid Computing is. If you don't, please read the Prerequisites documents section to get up to speed.

Furthermore, all these services are interacting constantly. For example, the Job Management Service might consult the Resource Discovery Service to find computational resources that match the job's requirements. With so many services, and so many interactions between them, there exists the potential for chaos. What if every vendor out there decided to implement a Job Management Service in a completely different way, exposing not only different functionality but also different interfaces? It would be very difficult (or nearly impossible) to get all the different software pieces to work together.

The solution is *standardization*: define a common interface for each type of service. For example, take a look at the World Wide Web. One of the reasons why the Web is such a popular Internet application is because it is based on *standards* (HTML, HTTP, etc.) agreed upon by all the different major players (Microsoft, Netscape, etc.). Imagine, on the other hand, that you could only use a Microsoft browser to access websites implemented with Microsoft technology (ditto for Netscape, Opera, etc.) It would be definitely uncool. Thanks to standards, I can use my favorite browser (provided it follows standards, which most modern browsers do) to access most of the websites out there (regardless of what technology is used to implement the website). Why? Because a set of common languages was agreed upon for all the browsers and websites out there. Standardization is definitely a good thing.

The Open Grid Services Architecture (OGSA), developed by The Global Grid Forum [http://www.ggf.org], aims to define a common, standard, and open architecture for grid-based applications. The goal of OGSA is to standardize practically all the services one commonly finds in a grid application (job management services, resource management services, security services, etc.) by specifying a set of standard interfaces for these services. At the time of writing this tutorial, this "set of standard interfaces" is still in the works. However, OGSA already defines a set of requirements that must be met by these standard interfaces. In other words, OGSA has already gone as far as identifying the most important services one encounters in Grid applications, and which most stand to benefit from standardization.

OGSA requires 'stateful services'

However, when the powers-that-be undertook the task of creating this new architecture, they realized they needed to choose some sort of distributed middleware on which to *base* the architecture. In other words, if OGSA (for example) defines that the JobSubmissionInterface has a submitJob method, there has to be a common and standard way to *invoke* that method if we want the architecture to be adopted as an industry-wide standard. This *base* for the architecture could, in theory, be any distributed middleware (CORBA, RMI, or even traditional RPC). For reasons that will be explained further on, Web Services were chosen as the underlying technology.

However, although the Web Services Architecture was certainly the best option, it still didn't meet one of OGSA's most important requirements: the underlying middleware had to be *stateful* (don't worry if you don't know what a "stateful" service is, it is explained in the next section). Unfortunately, although Web services can in theory be either stateless or stateful, they are usually stateless and there is no standard way of making them stateful. So, clearly, something had to be done!

WSRF

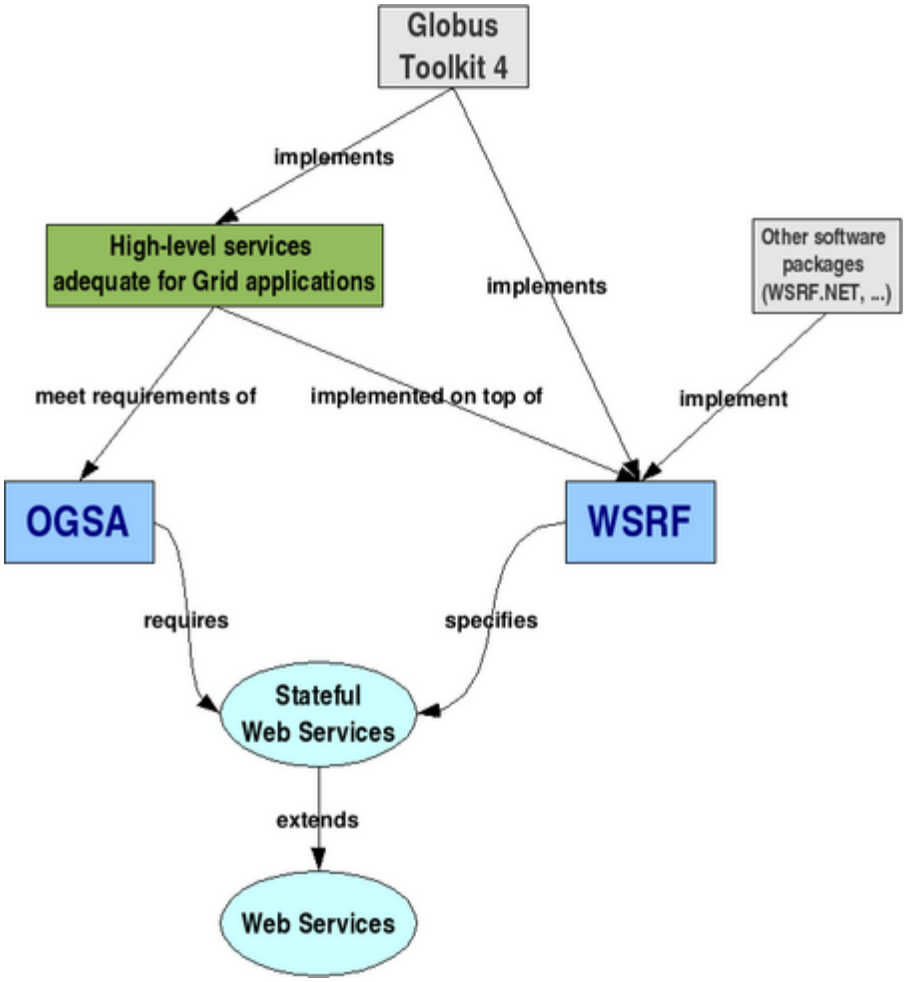
Enter the Web Services Resource Framework, a specification developed by OASIS [http://www.oasis-open.org]. WSRF specifies how we can make our Web Services stateful, along with adding a lot of other cool features. It is important to note that WSRF is a joint effort by the Grid and Web Services communities, so it fits pretty nicely inside the whole Web Services Architecture (in the diagram: *WSRF extends Web Services*).

So what exactly is the relation between OGSA and WSRF? It's very simple: WSRF provides the stateful services that OGSA needs. In the diagram: **WSRF specifies stateful services** (as opposed to those services simply 'being required' by OGSA). Another way of expressing this relation is that, while OGSA is the *architecture*, WSRF is the *infrastructure* on which that architecture is built on.

How does this relate to GT4?

Now that we've cleared up what OGSA and WSRF are, we are ready to complete the above diagram to see how GT4 fits into the picture:

Figure 1.2. Relationship between OGSA, GT4, WSRF, and Web Services



The Globus Toolkit 4

The Globus Toolkit is a software toolkit, developed by The Globus Alliance [http://www.globus.org], which we can use to program grid-based applications. The toolkit, first and foremost, includes quite a few *high-level services* that we can use to build Grid applications. These services, in fact, meet most of the abstract requirements set forth in OGSA. In other words, the Globus Toolkit includes a resource monitoring and discovery service, a job submission infrastructure, a security infrastructure, and data management services (to name a few!). Since the working groups at GGF are still working on defining standard interfaces for these types of services, we can't say (at this point) that GT4 is an implementation of OGSA (although GT4 does implement some security specifications defined by GGF). However, it *is* a realization of the OGSA requirements and a sort of *de facto* standard for the Grid community while GGF works on standardizing all the different services.

Most of these services are implemented *on top of WSRF* (the toolkit also includes some services that are not implemented on top of WSRF and are called the *non-WS components*). The Globus Toolkit 4, in fact, includes a complete implementation of the WSRF specification. This part of the toolkit (the WSRF implementation) is a very important part of the toolkit since nearly everything else is built on top of it. However, it's worth noting that it's also a *very small* part of the toolkit. At this point, we'll repeat something we said at the very beginning of the tutorial:

> At the end of this tutorial you will know how to program stateful Web Services using GT4. This will allow you to progress towards using the higher-level services of the toolkit. However it is important to understand that *you cannot program Grid-based applications using* only *the Java WS Core included in this tutorial*. This tutorial should be approached as a stepping stone towards more powerful tooling, not as a definite guide on GT4 programming.

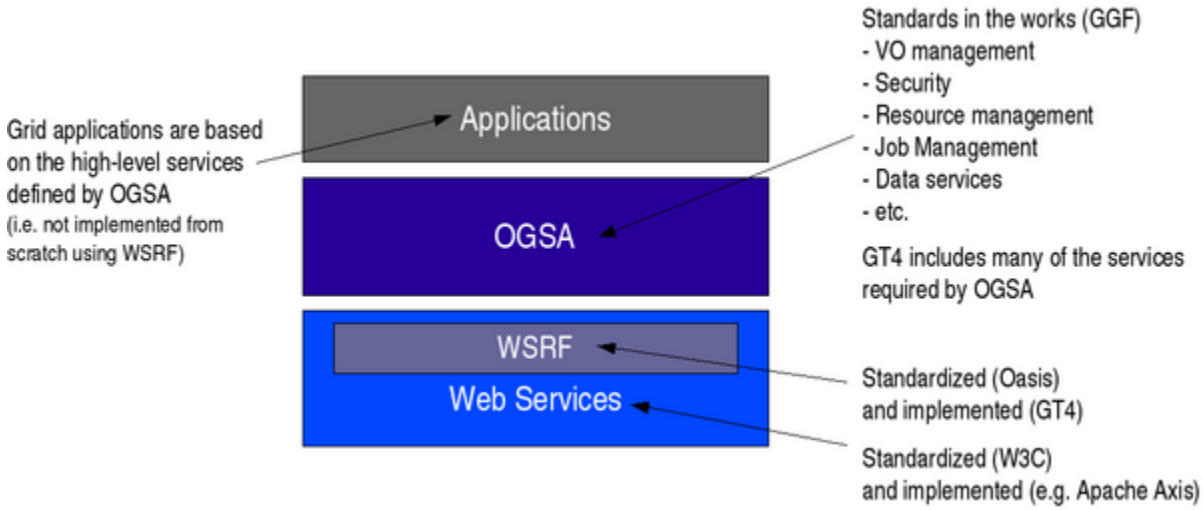
I'm sorry to insist so much on this, but this really is a very important concept. Never forget that, on the long road towards true Grid Nirvana, WSRF is indeed a necessary step, but only the *first* step.

Finally, take into account that GT4 isn't the only WSRF implementation out there. For example, another complete implementation of the WSRF specification is WSRF.NET [http://www.cs.virginia.edu/~gsw2c/wsrf.net.html].

The mandatory layered diagram

If there's something we computer geeks like in documentation, it's layered diagrams! So, this section really wouldn't be complete without a diagram that explains the relationship between OGSA, WSRF, and GT4 using the wonderful language of layers.

Figure 1.3. Layered diagram of OGSA, GT4, WSRF, and Web Services



A short introduction to Web Services

Before we take a closer look at what the Web Services Resource Framework (WSRF) is, we need to have a basic understanding of how Web Services work (so we can better appreciate how WSRF extends Web Services). If you're already familiar with Web Services, you can safely skip this section.

For quite a while now, there has been a lot of buzz about "Web Services," and many companies have begun to rely on them for their enterprise applications. So, what exactly are Web Services? To put it quite simply, they are *yet another* distributed computing technology (like CORBA, RMI, EJB, etc.). They allow us to create client/server applications.

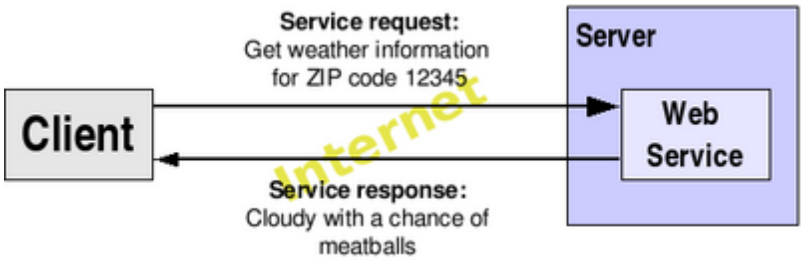
For example, let's suppose I keep a database with up-to-date information about weather in the United States, and I want to distribute that information to anyone in the world. To do so, I could *publish* the weather information through a Web Service that, given a ZIP code, will provide the weather information for that ZIP code.

Caution

Don't mistake this with publishing something on a *website*. Information on a website (like the one you're reading right now) is intended for humans. Information which is available through a Web Service will *always* be accessed by software, *never* directly by a human (despite the fact that there might be a human using that software). Even though Web Services rely heavily on existing Web technologies (such as HTTP, as we will see in a moment), they have no relation to web browsers and HTML. Repeat after me: websites for humans, Web Services for software :-)

The *clients* (programs that want to access the weather information) would then contact the *Web Service* (in the *server*), and send a *service request* asking for the weather information. The server would return the forecast through a *service response*. Of course, this is a very sketchy example of how a Web Service works. We'll see all the details in a moment.

Figure 1.4. Web Services



Some of you might be thinking: *"Hey! Wait a moment! I can do that with RMI, CORBA, EJBs, and countless other technologies!"* So, what makes Web Services special? Well, Web Services have certain advantages over other technologies:

- Web Services are platform-independent and language-independent, since they use standard XML languages. This means that my client program can be programmed in C++ and running under Windows, while the Web Service is programmed in Java and running under Linux.

- Most Web Services use HTTP for transmitting messages (such as the service request and response). This is a major advantage if you want to build an Internet-scale application, since most of the Internet's proxies and firewalls won't mess with HTTP traffic (unlike CORBA, which usually has trouble with firewalls).

Of course, Web Services also have some disadvantages:

- Overhead. Transmitting all your data in XML is obviously not as efficient as using a proprietary binary code. What you win in portability, you lose in efficiency. Even so, this overhead is usually acceptable for most applications, but you will probably never find a critical real-time application that uses Web Services.

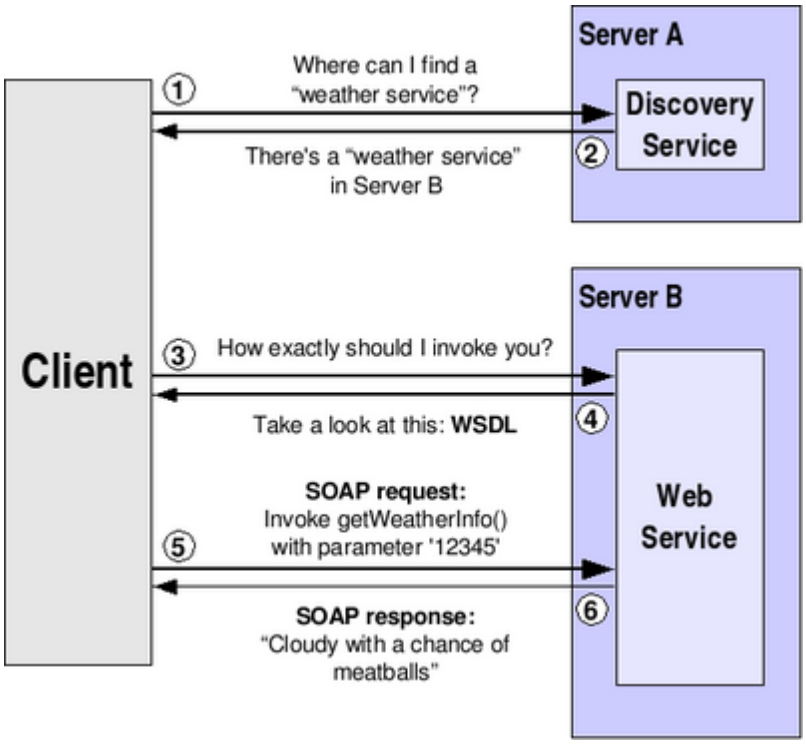
- Lack of versatility. Currently, Web Services are not very versatile, since they only allow for some very basic forms of service invocation. CORBA, for example, offers programmers a lot of supporting services (such as persistency, notifications, lifecycle management, transactions, etc.). Fortunately, there are a lot of emerging Web services specifications (including WSRF) that are helping to make Web services more and more versatile.

However, there is one important characteristic that distinguishes Web Services. While technologies such as CORBA and EJB are geared towards *highly coupled* distributed systems, where the client and the server are very dependent on each other, Web Services are more adequate for *loosely coupled* systems, where the client might have no prior knowledge of the Web Service until it actually invokes it. Highly coupled systems are ideal for intranet applications, but perform poorly on an Internet scale. Web Services, however, are better suited to meet the demands of an Internet-wide application, such as grid-oriented applications.

A Typical Web Service Invocation

So how does this all actually work? Let's take a look at all the steps involved in a complete Web Service invocation. For now, don't worry about all the acronyms (SOAP, WSDL, ...). We'll explain them in detail in just a moment.

Figure 1.5. A typical Web Service invocation



1. As we said before, a client may have no knowledge of what Web Service it is going to invoke. So, our first step will be to *discover* a Web Service that meets our requirements. For example, we might be interested in locating a public Web Service which can give me the weather forecast in US cities. We'll do this by contacting a *discovery service* (which is itself a Web service).

2. The discovery service will reply, telling us what servers can provide us the service we require.

3. We now know the location of a Web Service, but we have no idea of how to actually invoke it. Sure, we know it can give me the forecast for a US city, but how do we perform the actual service invocation? The method I have to invoke might be called "`string getCityForecast(int CityPostalCode)`", but it could also be called "`string getUSCityWeather(string cityName, bool isFarenheit)`". We have to ask the Web Service to *describe* itself (i.e. tell us how exactly we should invoke it)

4. The Web Service replies in a language called WSDL.

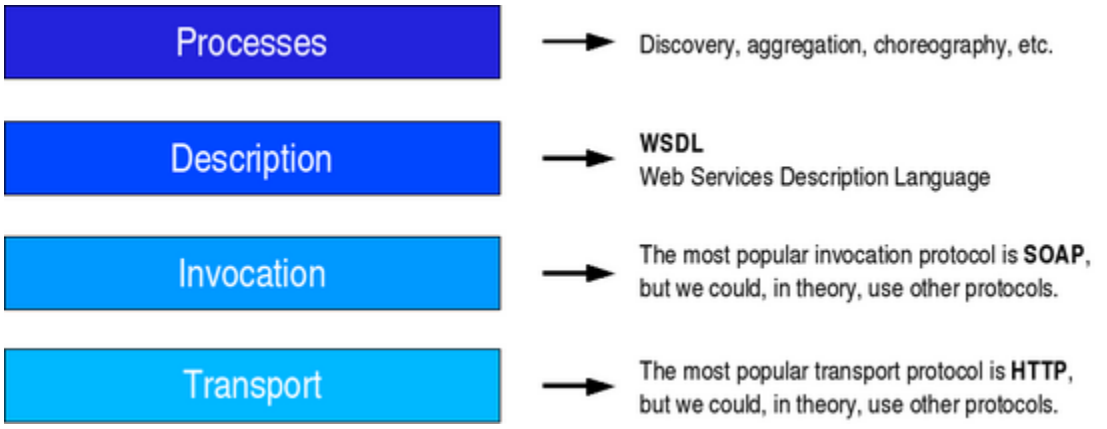
5. We finally know where the Web Service is located and how to invoke it. The invocation itself is done in a language called SOAP. Therefore, we will first send a *SOAP request* asking for the weather forecast of a certain city.

6. The Web Service will kindly reply with a *SOAP response* which includes the forecast we asked for, or maybe an error message if our SOAP request was incorrect.

Web Services Architecture

So, what exactly are SOAP and WSDL? They're essential parts of the Web Services Architecture:

Figure 1.6. The Web Services architecture



- **Service Processes**: This part of the architecture generally involves more than one Web service. For example, discovery belongs in this part of the architecture, since it allows us to locate one particular service from among a collection of Web services.
- **Service Description**: One of the most interesting features of Web Services is that they are *self-describing*. This means that, once you've located a Web Service, you can ask it to 'describe itself' and tell you what operations it supports and how to invoke it. This is handled by the Web Services Description Language (WSDL).
- **Service Invocation**: Invoking a Web Service (and, in general, any kind of distributed service such as a CORBA object or an Enterprise Java Bean) involves passing messages between the client and the server. SOAP (Simple Object Access Protocol) specifies how we should format requests to the server, and how the server should format its responses. In theory, we could use other service invocation languages (such as XML-RPC, or even some *ad hoc* XML language). However, SOAP is by far the most popular choice for Web Services.
- **Transport**: Finally, all these messages must be transmitted somehow between the server and the client. The protocol of choice for this part of the architecture is HTTP (HyperText Transfer Protocol), the same protocol used to access conventional web pages on the Internet. Again, in theory we could be able to use other protocols, but HTTP is currently the most used one.

In case you're wondering, most of the Web Services Architecture is specified and standardized by the World Wide Web Consortium [http://www.w3c.org/], the same organization responsible for XML, HTML, CSS, etc.

Web Services Addressing

We have just seen a simple Web Service invocation. At one point, a discovery service 'told' the client *where* the Web Service is located. But... how exactly are Web services addressed? The answer is very simple: just like web pages. We use plain and simple URIs (Uniform Resource Identifiers). If you're more familiar with the term URL (Uniform Resource Locator), don't worry: URI and URL are practically the same thing.

For example, the discovery registry might have replied with the following URI:

```
http://webservices.mysite.com/weather/us/WeatherService
```

This could easily be the address of a web page. However, remember that Web Services are always used by software (never directly by humans). If you typed a Web Service URI into your web browser, you

would probably get an error message or some unintelligible code (some web servers *will* show you a nice graphical interface to the Web Service, but that isn't very common). When you have a Web Service URI, you will usually need to give that URI to a program. In fact, most of the client programs we will write will expect to receive a Web service URI as a command-line argument.

Tip

If you're anxious to see a real Web service working, then today's your lucky day! A "Weather Web Service" is probably one of the most typical examples of a simple web service. You can find a real Weather Web Service here:

http://live.capescience.com/ccx/GlobalWeather

Wait a second... You didn't actually try to visit that URI, did you? Haven't you been paying attention? That's a Web service URI, so even though it may look and feel like the URIs you type in your browser when you want to visit your favorite website, this URI is meant only for software that "knows" how to invoke Web services.

Fortunately, the authors of that web service have been kind enough to provide a description [http://www.capescience.com/webservices/globalweather/index.shtml] of the Web service, along with a web interface [http://live.capescience.com/GlobalWeather] so you can actually invoke the service's methods. If you feel specially curious, you can even take a look at the Web service's WSDL [http://live.capescience.com/wsdl/GlobalWeather.wsdl] (also available in a slightly more readable version [http://www.w3.org/2000/06/webdata/xslt?xslfile=http://www.capescience.com/simplifiedwsdl.xslt&xmlfile=http://live.capescience.com/wsdl/GlobalWeather.wsdl&transform=Submit])

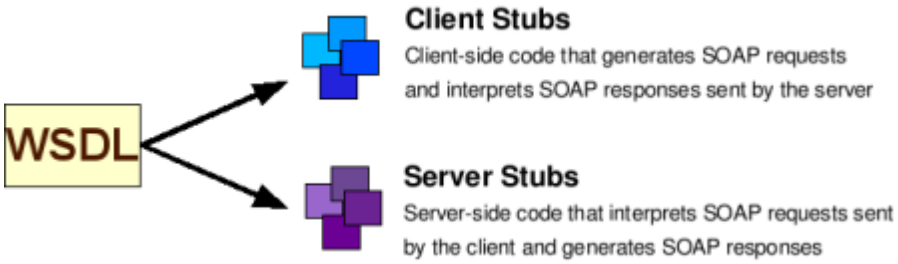
For example, if you visit the web interface [http://live.capescience.com/GlobalWeather] you'll see that the Weather Web service offers a `getWeatherReport` operation that expects a single string parameter (an IATA airport designation, e.g. ORD for Chicago O'Hare and LHR for London Heathrow). If you invoke `getWeatherReport`, the Web service will return a `WeatherReport` structure with all sorts of interesting weather data. Fun!

How does this work in practice?

OK, now that you have an idea of what Web Services are, you are probably anxious to start programming Web Services right away. Before you do that, you might want to know how Web Services-based applications are structured. If you've ever used CORBA or RMI, this structure will look pretty familiar.

First of all, you should know that despite having a lot of protocols and languages floating around, Web Services programmers usually only have to concentrate on writing code in their favorite programming language and, in some cases, in writing WSDL. SOAP code, on the other hand, is always generated and interpreted automatically for us. Once we've reached a point where our client application needs to invoke a Web Service, we *delegate* that task on a piece of software called a *stub*. The good news is that there are plenty of tools available that will generate stubs automatically for us, usually based on the WSDL description of the Web Service.

Figure 1.7. Client and server stubs are generated from the WSDL file



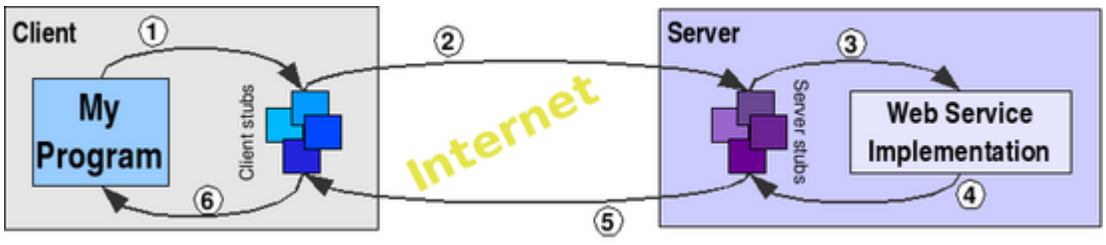
Using stubs simplifies our applications considerably. We don't have to write a complex client program that dynamically generates SOAP requests and interprets SOAP responses (and similarly for the server side of our application). We can simply concentrate on writing the client and/or server code, and leave all the dirty work to the stubs (which, again, we don't even have to write ourselves... they can be generated automatically from the WSDL description of a web service).

The stubs are generally generated only once. In other words, you shouldn't interpret the "Typical Web Service Invocation" figure (above) as saying that we go through the discovery process every single time we want to invoke a Web service, and generate the client stubs every time we want to invoke the service. In general, we only go through the discovery step once, then generate the stubs once (based on the WSDL of the service we've discovered) and then reuse the stubs as many times as we want (unless the maintainers of the Web service decide to change the service's interface and, thus, its WSDL description). Of course, there are more complex invocation scenarios, but for now the one we've described is more than enough to understand how Web services work.

A Typical Web Service Invocation (redux)

So, let's suppose that we've already located the Web Service we want to use (either because we consulted a discovery service, or because the Web service URI was given to us), and we've generated the client stubs from the WSDL description. What exactly happens when we want to invoke a Web service operation from a program?

Figure 1.8. A typical Web Service invocation (more detailed)



1. Whenever the client application needs to invoke the Web Service, it will really call the client stub. The client stub will turn this 'local invocation' into a proper SOAP request. This is often called the *marshaling* or *serializing* process.

2. The SOAP request is sent over a network using the HTTP protocol. The server receives the SOAP requests and hands it to the server stub. The server stub will convert the SOAP request into something the service implementation can understand (this is usually called *unmarshaling* or *deserializing*)

3. Once the SOAP request has been deserialized, the server stub invokes the service implementation, which then carries out the work it has been asked to do.

4. The result of the requested operation is handed to the server stub, which will turn it into a SOAP response.

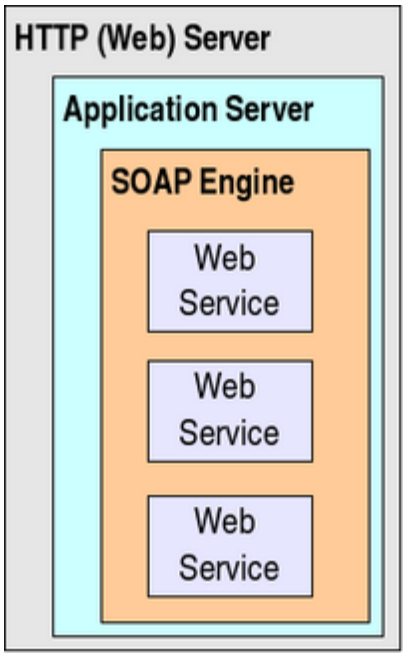
5. The SOAP response is sent over a network using the HTTP protocol. The client stub receives the SOAP response and turns it into something the client application can understand.

6. Finally the application receives the result of the Web Service invocation and uses it.

The server side, up close

Finally, let's take a close look at what the server looks like, specially what software we should expect to have to get Web services up and running on our server.

Figure 1.9. The server side in a Web Services application



- **Web service**: First and foremost, we have our Web service. As we have seen, this is basically a piece of software that exposes a set of operations. For example, if we are implementing our Web service in Java, our service will be a Java class (and the operations will be implemented as Java methods). Obviously, we want a set of clients to be able to invoke those operations. However, our Web service implementation knows nothing about how to interpret SOAP requests and how to create SOAP responses. That's why we need a...

- **SOAP engine**: This is a piece of software that knows how to handle SOAP requests and responses. In practice, it is more common to use a generic SOAP engine than to actually generate server stubs for each individual Web service (note, however, that we still need client stubs for the client). One good example of a SOAP engine is Apache Axis [http://ws.apache.org/axis/] (this is, in fact, the SOAP engine used by the Globus Toolkit). However, the functionality of the SOAP engine is usually limited to manipulating SOAP. To actually function as a server that can receive requests from different clients, the SOAP engine usually runs within an...

- **Application server**: This is a piece of software that provides a 'living space' for applications that must be accessed by different clients. The SOAP engine runs as an application inside the application server. A

good example is the Jakarta Tomcat [http://jakarta.apache.org/tomcat/] server, a Java Servlet and Java ServerPages container that is frequently used with Apache Axis and the Globus Toolkit.

Many application servers already include some HTTP functionality, so we can have Web services up and running by installing a SOAP engine and an application server. However, when an application server lacks HTTP functionality, we also need an...

- **HTTP Server**: This is more commonly called a 'Web server'. It is a piece of software that knows how to handle HTTP messages. A good example is the Apache HTTP Server [http://httpd.apache.org/], one of the most popular web servers in the Internet.

Note

Terminology in this area is still a bit inconsistent, so you might encounter different terms for the concepts we've just seen. In particular, it's very common to use the term *Web services container* as a catch-all term for the SOAP engine + application server + HTTP server.

WSRF: The Web Services Resource Framework

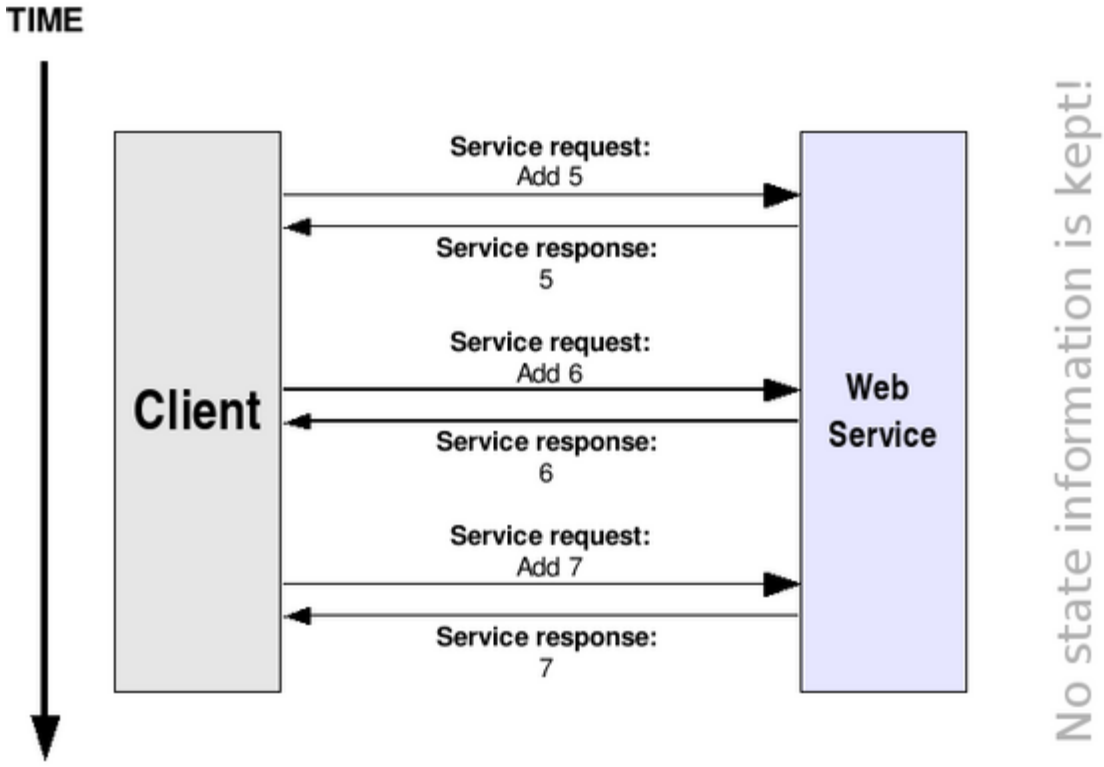
As we have just seen, Web Services are the technology of choice for Internet-based applications with loosely coupled clients and servers. That makes them the natural choice for building the next generation of grid-based applications. However, remember Web Services do have certain limitations. In fact, plain Web Services (as currently specified by the W3C) wouldn't be very helpful for building a grid application. Enter **WSRF**, which improves several aspects of web services to make them more adequate for grid applications.

In this section we'll take a brief look at the different parts of the WSRF specification. However, before doing that, we need to take a close look at the main improvement in WSRF: *statefulness*.

WSRF: It's all about state

Plain Web services are usually *stateless* (even though, in theory, there is nothing in the Web Services Architecture that says they can't be stateful). This means that the Web service can't "remember" information, or *keep state*, from one invocation to another. For example, imagine we want to program a very simple Web service which simply acts as an integer accumulator. This accumulator is initialized to zero, and we want to be able to add (accumulate) values in it. Suppose we have an `add` operation which receives the value to add and returns the current value of the accumulator. As shown in the following figure, our first invocation of this operation might seem to work (we request that 5 be added, and we receive 5 in return). However, since a Web service is stateless, the following invocations have no idea of what was done in the previous invocations. So, in the second call to `add` we get back 6, instead of 11 (which would be the expected value if the Web service was able to keep state).

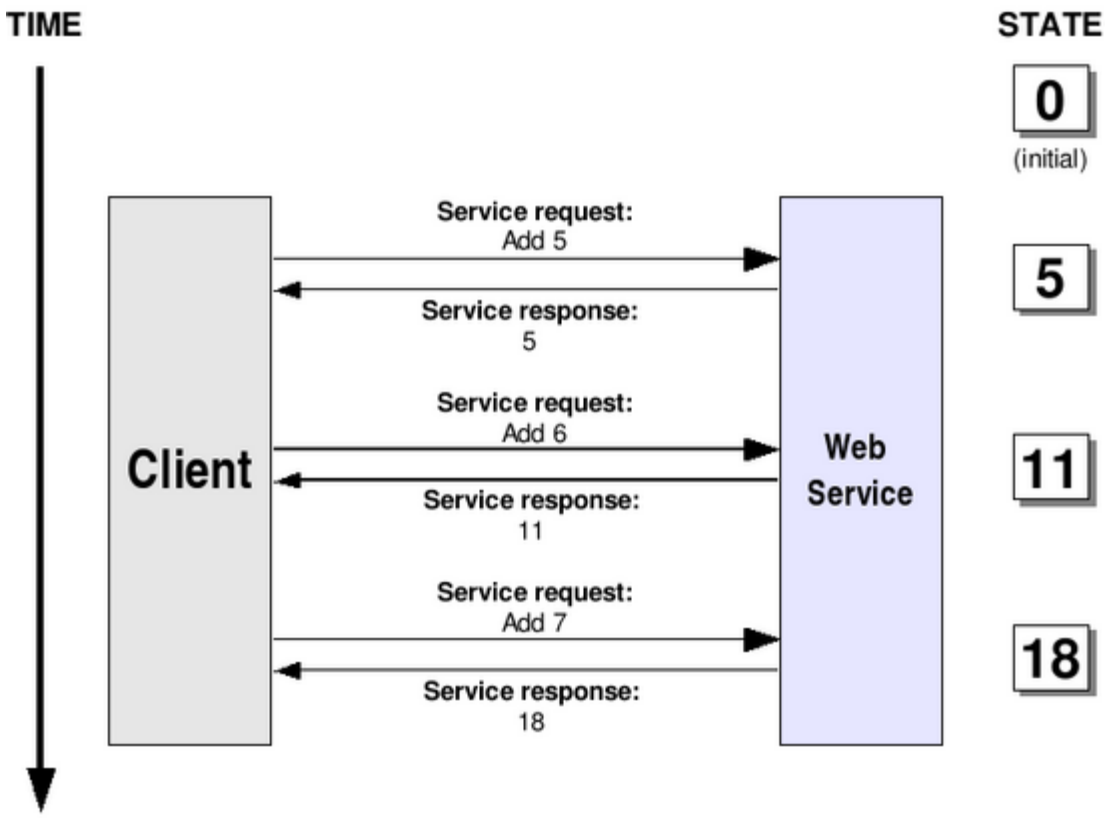
Figure 1.10. A stateless Web Service invocation



The fact that Web services don't keep state information is not necessarily a bad thing. There are plenty of applications which have no need whatsoever for statefulness. For example, the Weather Web service we saw in the previous section is a real, working Web service which has no need to know what happened in the previous invocations.

However, Grid applications *do* generally require statefulness. So, we would ideally like our Web service to somehow keep state information:

Figure 1.11. A stateful Web Service invocation



However, this is a pretty peculiar dilemma since, as mentioned above, a Web service is usually a stateless entity. In fact, some people might argue that a "stateful Web service" is a bit of a contradiction in terms! So, how do we get out of this jam?

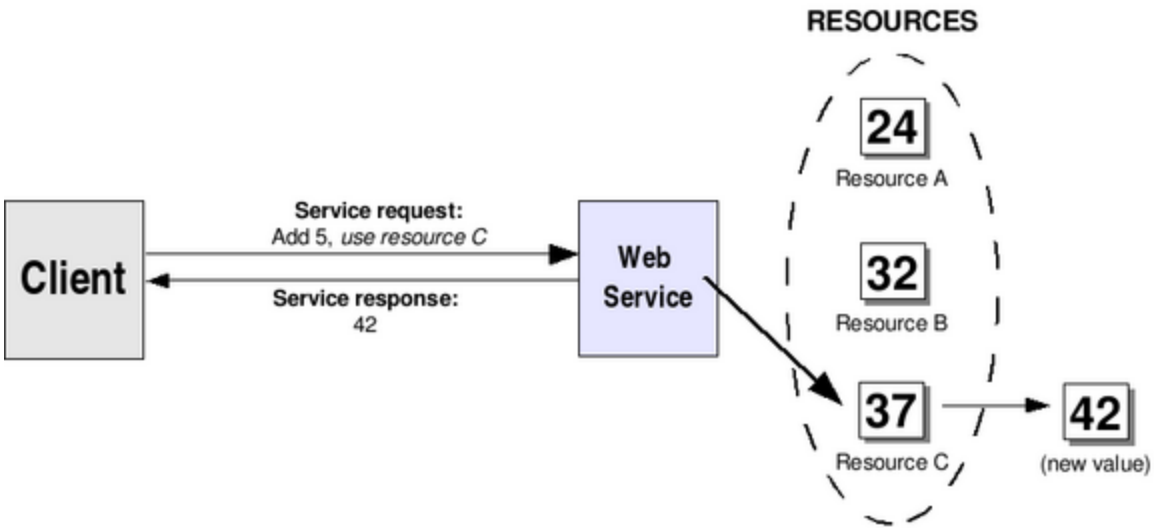
The resource approach to statefulness

Giving Web services the ability to keep state information while still keeping them stateless seems like a complex problem. Fortunately, it's a problem with a very simple solution: simply keep the Web service and the state information completely separate.

Instead of putting the state *in* the Web service (thus making it stateful, which is generally regarded as a bad thing) we will keep it in a separate entity called a *resource*, which will store all the state information. Each resource will have a unique *key*, so whenever we want a *stateful interaction* with a Web service we simply have to instruct the Web service to use a particular resource.

For example, take the accumulator example. As shown in the next figure, our Web service could have three different resources (A, B, C) to choose from. If we want the integer value to be 'remembered' from invocation to invocation, the client simply has to specify that he wants a method invoked *with* a certain resource.

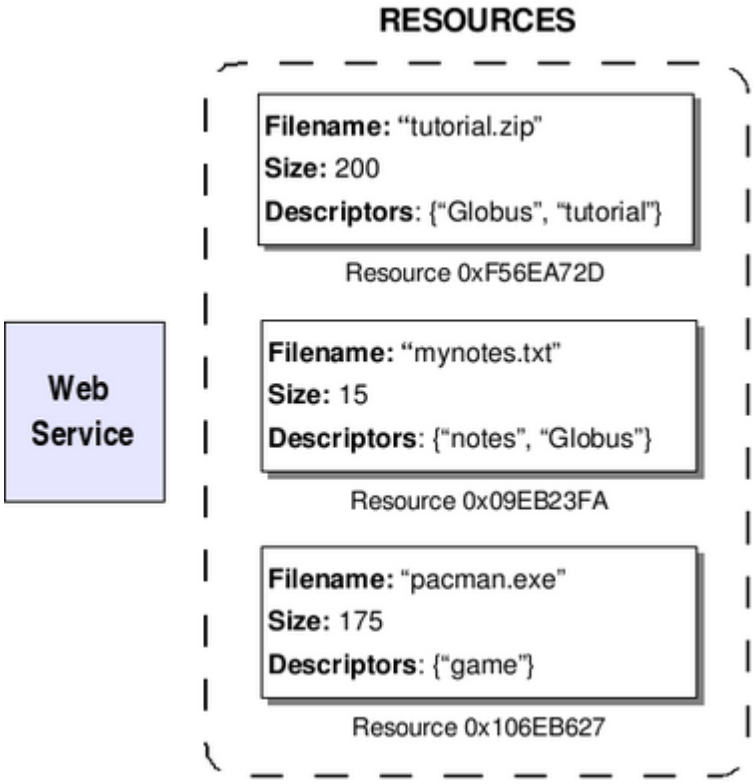
Figure 1.12. The resource approach to statefulness



In the figure we can see that the client wants the `add` operation invoked with resource C. When the Web service receives the `add` request, it will make sure to retrieve resource C so that `add` is actually performed on that resource. The resource themselves can be stored in memory, on secondary storage, or even in a database. Also, notice how a Web service can have access to more than one resource.

Of course, resources can come in all different shapes and sizes. A resource can keep multiple values (not just a simple integer value, as shown in the previous figure). For example, our resources could represent files:

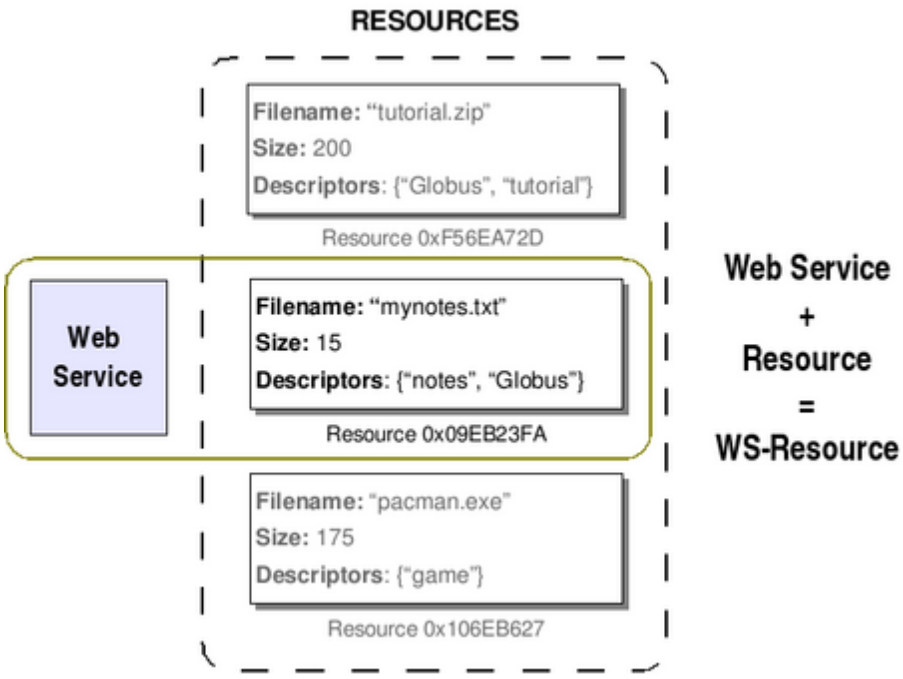
Figure 1.13. A Web Service with several resources. Each resource represents a file.



You might be wondering: And how exactly does the client specify what resource must be used? A URI might be enough to address the Web service, but how do we specify the resource on top of that? There are actually several different ways of doing this. As we'll see later on, the preferred way of doing it is to use a relatively new specification called WS-Addressing which provides a more versatile way of addressing Web Services (when compared to plain URIs).

Finally, a bit of terminology before we continue. A pairing of a Web service with a resource is called a WS-Resource. The address of a particular WS-Resource is called an *endpoint reference* (this is WS-Addressing lingo).

Figure 1.14. WS-Resource



The WSRF specification

The Web Services Resources Framework is a collection of five different specifications. Of course, they all relate (in some way or another) to the management of WS-Resources.

WS-ResourceProperties

A resource is composed of zero or more *resource properties*. For example, in the figure shown above each resource has three resource properties: Filename, Size, and Descriptors. WS-ResourceProperties specifies how resource properties are defined and accessed. As we'll see later on when we start programming, the resource properties are defined in the Web service's WSDL interface description.

WS-ResourceLifetime

Resources have non-trivial lifecycles. In other words, they're not a static entity that is created when our server starts and destroyed when our server stops. Resources can be created and destroyed at any time. The WS-ResourceLifetime supplies some basic mechanisms to manage the lifecycle of our resources.

WS-ServiceGroup

We will often be interested in managing *groups of Web Services* or *groups of WS-Resources*, and performing operations such as 'add new service to group', 'remove this service from group', and (more importantly) 'find a service in the group that meets condition FOOBAR'. The WS-ServiceGroup specifies how exactly we should go about grouping services or WS-Resources together. Although the functionality provided by this specification is very basic, it is nonetheless the base of more powerful discovery services (such as GT4's IndexService) which allow us to group different services together and access them through a single point of entry (the service group).

WS-BaseFaults

Finally, this specification aims to provide a standard way of reporting faults when something goes wrong during a WS-Service invocation.

Related specifications

WS-Notification

WS-Notification is another collection of specifications that, although not a part of WSRF, is closely related to it. This specification allows a Web service to be configured as a *notification producer*, and certain clients to be *notification consumers* (or subscribers). This means that if a change occurs in the Web service (or, more specifically, in one of the WS-Resources), that change is *notified* to all the subscribers (not *all* changes are notified, only the ones the Web services programmer wants to).

WS-Addressing

As mentioned before, the WS-Addressing specification provides us a mechanism to address Web services which is much more versatile than plain URIs. In particular, we can use WS-Addressing to address a Web service + resource pair (a WS-Resource).

The Globus Toolkit 4

So, WSRF sure seems pretty cool and exciting, huh? However, if you've already programmed Grid-based applications, you're probably thinking that this is all very nice, but hardly enough for The Grid. Remember that WSRF is only a small (but important!) part of the whole GT4 Architecture: it is the *infrastructure* on top of which most of the toolkit is built. Besides the WSRF implementation, the toolkit includes a lot of components which we can use to program Grid applications.

Architecture

The Globus Toolkit 4 is composed of several software components. As shown in the following figure, these components are divided into five categories: Security, Data Management, Execution Management, Information Services, and the Common Runtime. Notice how, despite the fact that GT4 focuses on Web services, the toolkit also includes components which are *not* implemented on top of Web services. For example, the GridFTP component uses a non-WS protocol which started as an ad hoc Globus protocol, but later became a GGF specification.

Figure 1.15. GT4 architecture

As mentioned in the preface, the tutorial currently focuses only on the Java WS Core component. Once again, it is important to realize that the Globus Toolkit includes a lot of other components which can help us build Grid systems. Even so, the Java WS Core component is specially interesting because it is the base for most of the WS components. Note that we do not need to have in-depth knowledge about Java WS Core to *use* many GT4 components like GRAM, MDS, etc. However, if we want to build a Grid system that *integrates* all of these components with our own services, we will need to know about Java WS Core to actually "glue" all those services together and to program our own services.

GT4 Components

Let's take a quick look at what we can find in each of the five families of GT4 component. For more detailed descriptions, please refer to the official Globus documentation or to the Globus Primer (you can find a link in the preface)

Common Runtime

The Common Runtime components provide a set of fundamental libraries and tools which are needed to build both WS and non-WS services.

Security

Using the Security components, based on the Grid Security Infrastructure (GSI), we can make sure that our communications are secure.

Data management

These components will allow us to manage large sets of data in our virtual organization.

Information services

The Information Services, more commonly referred to as the Monitoring and Discovery Services (MDS), includes a set of components to discover and monitor resources in a virtual organization. Note that GT4 also includes a non-WS version of MDS (MDS2) for legacy purposes. This component is deprecated and will surely disappear in future releases of the toolkit.

Execution management

Execution Management components deal with the initiation, monitoring, management, scheduling and coordination of executable programs, usually called *jobs*, in a Grid.

Where to learn Python & XML

After seeing all the theory behind GT4, we're almost ready to start programming. However, you need to know Python to follow this tutorial.

- The Python HomePage [http://www.python.org]

Also, you need to be familiar with XML. You don't have to be an XML wizard, but should at least be able to read and interpret the different elements of an XML document. If you've never worked with XML, you should probably take a look at the following sites:

- W3Schools XML Tutorial [http://www.w3schools.com/xml/] : Tutorial that covers both the basics and the more advanced aspects of XML.

- ZVON.org [http://www.zvon.org/] : Tons of XML resources. Includes some very good reference guides.

Chapter 2. Installation

Required Dependencies:

- Python >= 2.4 [http://www.python.org]
- pyXML >= 0.8.4 [http://pyxml.sourceforge.net/]
- 4Suite-XML >= 1.0.2 [http://www.4suite.org]
- ZSI >= 2.1.dev_r1367 [http://sourceforge.net/projects/pywebsvcs]
- Twisted >= 2.4.0 [http://www.twistedmatrix.com/products/download]
  - TwistedCore >= 2.4.0
  - TwistedWeb >= 0.6.0
- Zope Interfaces 3.0.1 [http://zope.org/Products/ZopeInterface]

Optional Security Dependencies

- HTTPS Transport Security M2Crypto >= 0.17 [http://sandbox.rulemaker.net/ngps/m2/]

  ### Caution

  Please see the README for information on installing

- GSI Message Security pyGsi >= 0.2 [http://dsd.lbl.gov/gtg/projects/pyGridWare/pyGsi.html]

  ### Caution

  Please see the README for information on installing

Download:

- pyGridWare [http://dev.globus.org/wiki/PyGridWare]

Installation:

- Please See INSTALL for directions

Part II. GT4 Core

Table of Contents

Chapter 3. Writing a NON-WSRF Stateful Web Service

MathService

In this chapter we are going to write and deploy a simple stateful web service. This service *does not* use WSRF to keep state information, we'll explore that in the next chapter. Our first web service is an extremely simple calculator, which we'll refer to as *MathService*. It will provide the following operations:

- Add an integer value to the stored value
- Subtract an integer value from the stored value
- Retrieve current stored (integer) value

MathService's internal logic is very simple. When it, a new resource, is created, its stored integer value is initialized to zero. The addition and subtraction operations modify this stored value.

Finally, this first example will be limited to having only one instance, or *resource*. In the next chapter we will see how we can write a WSRF service that has several resources associated with it, as seen in ????.

High-tech stuff, huh? Don't worry if this seems a bit lackluster. Since this is going to be our first stateful web service, it's better to start with a small didactic service which we'll gradually improve by adding more complex resource properties, notifications, etc.

The Four Steps

Writing and deploying a WSRF Web Service is easier than you might think. You just have to follow five simple steps.

1. **Define the service's interface**. This is done with *WSDL*.
2. **Create a New WSRF Site**. This is done with *wsdl2web* script.
3. **Implement the service functionality**. This is done with *Python*.
4. **Deploy the service**. This is done with *Python*.

In this first example we're going to go through each step in great detail, explaining what each step accomplishes, and giving detailed instructions on how to perform each step. The rest of the examples in the tutorial will also follow these four steps, but won't repeat the whole explanation of what each step is. So, if you ever find that you don't understand a particular step, you can always come back to this chapter ("Writing Your First Stateful Web Service in 4 Simple Steps") to review the details of that step.

Step 1: Defining the interface in WSDL

The first step in writing any web service is to define the *service interface*. We need to specify what our service is going to provide to the outer world. At this point we're not concerned with the inner workings of that service (what algorithms it uses, other systems it interacts with, etc.). We just need to know what

operations will be available to our users. In Web Services lingo, the service interface is usually called the *port type* (usually written *portType*, to match the XML element name).

As we saw in ????, there is a special XML language which can be used to specify what operations a web service offers: the Web Service Description Language (WSDL). So, what we need to do in this step is write a description of our MathService using WSDL.

WSDL can be automatically generated from other interface languages, or source code. But at this time, these methods still tend to be buggy and generate overly complex WSDL files. In addition, for a dynamically typed language like Python, it is difficult to generate WSDL from the class definitions. For these and didactic purposes, we will start by generating the WSDL directly.

The goal of this page is not to give a detailed explanation of how to write a WSDL file, but rather to present the WSDL file for this particular example. If you have no idea whatsoever of how to write WSDL, now is a good time to take a look at ????. Come on, go take a look at the appendix. We'll be waiting for you right here.

The WSDL code

Ok, so supposing you either know WSDL or have visited the WSDL appendix, take a good thorough look at this WSDL code. Download SimpleMath.wsdl [http://dsd.lbl.gov/gtg/projects/pyGridWare/doc/downloads/v1.4.0/simple/SimpleMath.wsdl]

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="MathService"

 targetNamespace="http://www.globus.org/namespaces/examples/core/
MathService_instance"
    xmlns="http://schemas.xmlsoap.org/wsdl/"

 xmlns:tns="http://www.globus.org/namespaces/examples/core/
MathService_instance"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"

 xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties-1.2-draft-01.xsd"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">


<!--============================================================

                     T Y P E S

  ============================================================-->
<types>
<xsd:schema
 targetNamespace="http://www.globus.org/namespaces/examples/core/
MathService_instance"

 xmlns:tns="http://www.globus.org/namespaces/examples/core/
MathService_instance"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
```

```
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">

        <!-- IMPORT WS-ADDRESSING -->
        <xsd:import
 namespace="http://schemas.xmlsoap.org/ws/2004/03/addressing"

 schemaLocation="http://schemas.xmlsoap.org/ws/2004/03/addressing"/>

 <!-- REQUESTS AND RESPONSES -->

 <xsd:element name="add" type="xsd:int"/>
 <xsd:element name="addResponse">
  <xsd:complexType/>
 </xsd:element>

 <xsd:element name="subtract" type="xsd:int"/>
 <xsd:element name="subtractResponse">
  <xsd:complexType/>
 </xsd:element>

 <xsd:element name="getValueRP">
  <xsd:complexType/>
 </xsd:element>
 <xsd:element name="getValueRPResponse" type="xsd:int"/>

        <!-- UNCOMMENT FOR CHAPTER 4 FACTORY METHOD
 <xsd:element name="create">
  <xsd:complexType/>
 </xsd:element>
 <xsd:element name="createResponse">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element ref="wsa:EndpointReference"/>
                </xsd:sequence>
            </xsd:complexType>
 </xsd:element>
        UNCOMMENT FOR CHAPTER 4 FACTORY METHOD -->

        <!-- UNCOMMENT FOR CHAPTER 4 FACTORY METHOD RESOURCE
 PROPERTIES
        <xsd:element name="Value" type="xsd:int"/>
        <xsd:element name="LastOp" type="xsd:string"/>
        <xsd:element name="MathResourceProperties">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element ref="tns:Value"/>
                    <xsd:element ref="tns:LastOp"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        UNCOMMENT FOR CHAPTER 4 FACTORY METHOD RESOURCE PROPERTIES -->

</xsd:schema>
</types>
```

```
<!--============================================================

                     M E S S A G E S

  ============================================================-->
<message name="AddInputMessage">
 <part name="parameters" element="tns:add"/>
</message>
<message name="AddOutputMessage">
 <part name="parameters" element="tns:addResponse"/>
</message>

<message name="SubtractInputMessage">
 <part name="parameters" element="tns:subtract"/>
</message>
<message name="SubtractOutputMessage">
 <part name="parameters" element="tns:subtractResponse"/>
</message>

<message name="GetValueRPInputMessage">
 <part name="parameters" element="tns:getValueRP"/>
</message>
<message name="GetValueRPOutputMessage">
 <part name="parameters" element="tns:getValueRPResponse"/>
</message>

<!-- UNCOMMENT FOR CHAPTER 4 FACTORY METHOD
<message name="CreateInputMessage">
 <part name="parameters" element="tns:create"/>
</message>
<message name="CreateOutputMessage">
 <part name="parameters" element="tns:createResponse"/>
</message>
UNCOMMENT FOR CHAPTER 4 FACTORY METHOD -->

<!--============================================================

                     P O R T T Y P E

  ============================================================-->

<portType name="MathPortType"
>
<!-- REMOVE PREVIOUS > & UNCOMMENT FOR CHAPTER 4 FACTORY METHOD
    wsrp:ResourceProperties="tns:MathResourceProperties">
UNCOMMENT FOR CHAPTER 4 FACTORY METHOD -->

 <operation name="add">
  <input message="tns:AddInputMessage"/>
  <output message="tns:AddOutputMessage"/>
 </operation>
```

```
 <operation name="subtract">
  <input message="tns:SubtractInputMessage"/>
  <output message="tns:SubtractOutputMessage"/>
 </operation>

 <operation name="getValueRP">
  <input message="tns:GetValueRPInputMessage"/>
  <output message="tns:GetValueRPOutputMessage"/>
 </operation>

        <!-- UNCOMMENT FOR CHAPTER 4 FACTORY METHOD
 <operation name="createResource">
  <input message="tns:CreateInputMessage"/>
  <output message="tns:CreateOutputMessage"/>
 </operation>
        UNCOMMENT FOR CHAPTER 4 FACTORY METHOD -->


</portType>

<!--================================================================

                        B I N D I N G

  ================================================================-->
<binding name="MathBinding" type="MathPortType">
    <soap:binding style="document"
 transport="http://schemas.xmlsoap.org/soap/http"/>

    <wsdl:operation name="add">
      <soap:operation soapAction=""/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>

    <wsdl:operation name="subtract">
      <soap:operation soapAction=""/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>

    <wsdl:operation name="getValueRP">
      <soap:operation soapAction=""/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
```

```
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>

    <!-- UNCOMMENT FOR CHAPTER 4 FACTORY METHOD
    <wsdl:operation name="createResource">
      <soap:operation soapAction=""/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    UNCOMMENT FOR CHAPTER 4 FACTORY METHOD -->
</binding>

<!--============================================================

                      S E R V I C E

  ============================================================-->
<service name="MathService">
    <port name="MathPort" binding="MathBinding">
       <soap:address location="http://localhost:8080/wsrf/services/"/>
    </port>
</service>

</definitions>
```

If you know WSDL, you'll recognize this as a pretty straightforward WSDL file which defines three operations: `add`, `subtract`, and `getValueRP` (along with all the necessary messages and types). By the way, the "RP" in `getValueRP` stands for "Resource Property" because, in a Web Services terminology, the integer value is a *property* of the MathService *resource*.

Step 2: Create a New WSRF Site

Now that we have defined the service's interface, we need to generate the Python skeleton of that interface. This skeleton will need to know what its own name is and what types of messages to accept, in this sense *binding* the abstract interface to a particular runnable instance. In order to do this, you will need to run the Python `wsdl2web` script, which will set up a new site.

wsdl2web

The WSDL presented above available at SimpleMath.wsdl [http://dsd.lbl.gov/gtg/projects/pyGridWare/doc/downloads/v1.4.0/simple/SimpleMath.wsdl]

Create the new Site.

```
% mkdir /tmp/SimpleMathSite; cd /tmp/SimpleMathSite
% wsdl2web --script=client.py SimpleMath.wsdl
```

```
pyGridWare/utility/generate/Utility.py:47:
UserWarning: portType(MathPortType) does not have a ResourceProperties

declaration (1)
```

(1) This warning is informing us that there is no WSRP ResoureProperties declaration, this will be further elaborated on.

After the script has finished there will be several new files:

```
% ls
README (1)
SimpleMath.wsdl (2)
client.py (3)
generated (4)
twisted (5)
```

(1) descriptions of site's files

(2) Original SimpleMath WSDL file

(3) Barebones sample client script

(4) Directory with all code generated from the WSDL

(5) twisted plugins directory, contains plugins for several twistd commands (wsrf, wsrf-ssl, wsrf-gsi)

The generated service stub contained in this file `generated/SimpleMath/services/SimpleMath/MathService.py` contains SOAP binding information and is used to marshall data between XML and python types. It doesn't add or subtract integers.

Step 3: Implement the service

After generating the site, we need to actually provide the implementation of the service's advertised functionality. Recall that in the example WSDL we defined one service, called *MathService*. The generation step created a service skeleton stub named `MathServiceWSRF` Now we will create a new class that subclasses this generated service skeleton stub and overrides all the methods prefixed with the special 4-character sequence "wsa_". One of these methods should be present for each "operation" defined in the WSDL file. Put another way, these methods represent the WSDL operations of the service. For example, "wsa_add" represents the service's "add" operation.

To understand the code you can see in these methods, we must first understand something about Web Service *request* and *response* objects.

- *request*: When a "wsa_"-prefixed method of the *generated service skeleton stub* is invoked (e.g. `MathServiceWSRF.wsa_add()`), it creates a new request instance and returns it. This request represents the XML message that was passed in the `ps` parameter ("ps" stands for "parsed SOAP", i.e. a parsed web services message). This Python object will have an attribute called `typecode` which is used to serialize it back into XML.

- *response*: When a "wsa_"-prefixed method of the *generated service skeleton stub* returns, it creates a new response instance. This object also contains a `typecode` attribute, which is used to serialize it into XML.

The important thing to get out of the above is that both requests and responses are objects with the attribute "typecode". They could be subclasses of integers or strings or floating-point numbers that just have an extra "typecode" attribute tagging along for the ride. In fact, with this simple service this is exactly the case, and understanding that helps understand some details of the code.

Save this file somewhere in your PYTHONPATH SimpleMathService.py [http://dsd.lbl.gov/gtg/projects/pyGridWare/doc/downloads/v1.4.0/simple/SimpleMathService.py]

```
from generated.SimpleMath.services.SimpleMath.MathService import
 MathServiceWSRF

class StatefulWebService(MathServiceWSRF):

    def __init__(self, *args, **kw):
        MathServiceWSRF.__init__(self, *args, **kw)
        self.value = 0❶


    def wsa_add(self, ps, address, **kw):❷
        request,response = MathServiceWSRF.wsa_add(self, ps, address,
 **kw)❸
        self.value += request❹
        return request,response

    def wsa_subtract(self, ps, address, **kw):
        request,response = MathServiceWSRF.wsa_subtract(self, ps,
 address, **kw)
        self.value -= request
        return request,response

    def wsa_getValueRP(self, ps, address, **kw):
        request,response = MathServiceWSRF.wsa_getValueRP(self, ps,
 address, **kw)
        return request,response.__class__(self.value)❺
```

❶ The attribute (*self.value*) is used to save the state.

❷ "wsa_" prefixed methods are expected to take these parameters

❸ Each wsa method is expected to return a python object representing the output message of the corresponding WSDL operation. The base class returns an appropriate instance to us.

❹ The base class returns a python object representing the input message of the corresponding WSDL operation, this is the *request* python instance. For the add operation the request is an integer type element, so the request can be added to *self.value*.

❺ This one is a little tricky. The response object is just an integer with an additional "typecode" attribute (see request/response discussion above). Integers are immutable, so in order to set the proper value, *self.value*, a new instance must be created that contains this value. To do this, we get the class of the response with *response.__class__*, and provide the integer *self.value* to the class constructor.

Step 4: Deploy the service

There are several ways to deploy a service. In this case we are using a stateful web service so we'll want to create a single instance and deploy it at a fixed URL. We are using here the web infrastructure provided by the popular "Twisted" framework. [Note: Twisted does not come with Python, so you will need to first download and install it from twistedmatrix.com [http://twistedmatrix.com/]. You will need the core library and also the "Twisted Web" server.]

First, let's append a setup function to the `SimpleMathService.py` file.

```
############################################################################
# SetUpResourceTree
#
############################################################################
def SetUpResourceTree(root):❶
    from twisted.web.resource import Resource
    root.putChild('ws', Resource())
    root.getStaticEntity('ws').putChild('MathService',
 StatefulWebService())
```

This function takes a single parameter `root`, this represents the `/` resource

Next, we will run the `wsrf twistd`command, and direct the command through a command line option to call this setup function.

```
twistd -no wsrf --tree=SimpleMathService.SetUpResourceTree
```

Congratulations! You've just created and deployed a stateful Web Service! But how do you know it really worked? In the next section, we'll write a simple client to test our new service.

Barebones MathService client.

In this section, we'll quickly show you how to write a client to test out your newborn stateful Web Service. The client will have minimal error-checking, but should work fine just connecting on your local host. One important thing to check before you start is the firewall setup on your computer and make sure that the firewall allows connections to/from your localhost (127.0.0.1). You do not need to open any holes in your external connectivity firewall for this client to work.

- Add the three footnoted lines below to *client.py* and copy it to *SimpleMathClient.py* . Or download SimpleMathClient.py [http://dsd.lbl.gov/gtg/projects/pyGridWare/doc/downloads/v1.4.0/simple/SimpleMathClient.py] .

```
############################################################################
# Automatically generated by wsdl2web.py
# See LBNLCopyright for copyright notice!
############################################################################
from twisted.python import log
from twisted.internet import reactor

import ZSI
```

```
from pyGridWare.utility.scripts.client import GetBasicOptParser,\
    GetPortKWArgs, SetUp
from generated.SimpleMath.stubs import SimpleMath as CLIENT


def main(**kw):
    locator = CLIENT.MathServiceLocator()
    port = locator.getMathPort(**kw)

    port.add(CLIENT.AddInputMessage(10)) ❶
    port.subtract(CLIENT.SubtractInputMessage(3)) ❷
    print "VALUE: ",
 port.getValueRP(CLIENT.GetValueRPInputMessage()) ❸

    # Factory METHOD Just guessing here
    #response = port.create(CLIENT.CreateRequest())
    #kw['endPointReference'] = response._EndpointReference
    #iport = locator.getMathPort(**kw)
    reactor.stop()


if __name__ == '__main__':
    op = GetBasicOptParser()
    (options, args) = op.parse_args()
    SetUp(options)
    kw = GetPortKWArgs(options)
    reactor.callWhenRunning(main, **kw)
    reactor.run()
```

❶ Invoke the `add` method, it takes an `AddInputMessage` instance, it is an integer.

❷ Invoke the `subtract` method, it takes a `SubtractInputMessage` instance, it is an integer.

❸ Retrieve the counter value by invoking the `getValueRP` method, the `GetValueRPInputMessage` doesn't have any contents.

- You should see the following output. If you want to see the soap messages send debugging information to stdout by using `-d 1`

```
%./SimpleMathClient.py -u http://127.0.0.1:9080/ws/MathService -d 0
VALUE:  7
```

Chapter 4. Writing a WS-RF Service with Multiple Resources

In the previous chapters we implemented a simple stateful web service that used local attributes to keep stateful information.

In this chapter we will learn how to write a service that, using a design pattern known as the *factory* pattern, will be able to manage *multiple* resources. To accomplish this we'll specify our service's state through resource properties and manage that state through WSRF specific mechanisms.

The WS-Resource factory pattern

The *factory* pattern is a well-known design pattern in software design, especially in object-oriented languages (Wikipedia entry for factory pattern [http://en.wikipedia.org/wiki/Abstract_factory_pattern]). In this pattern, we are not allowed to create instances of objects directly, but must do so through a *factory* that will provide a `create` operation.

When dealing with multiple resources, the WSRF specs recommend that we follow this pattern, having one service in charge of creating the resources (*"the factory service"*) and another one to actually access the information contained in the resources (*"the instance service"*).

Figure 4.1. The WS-Resource factory pattern

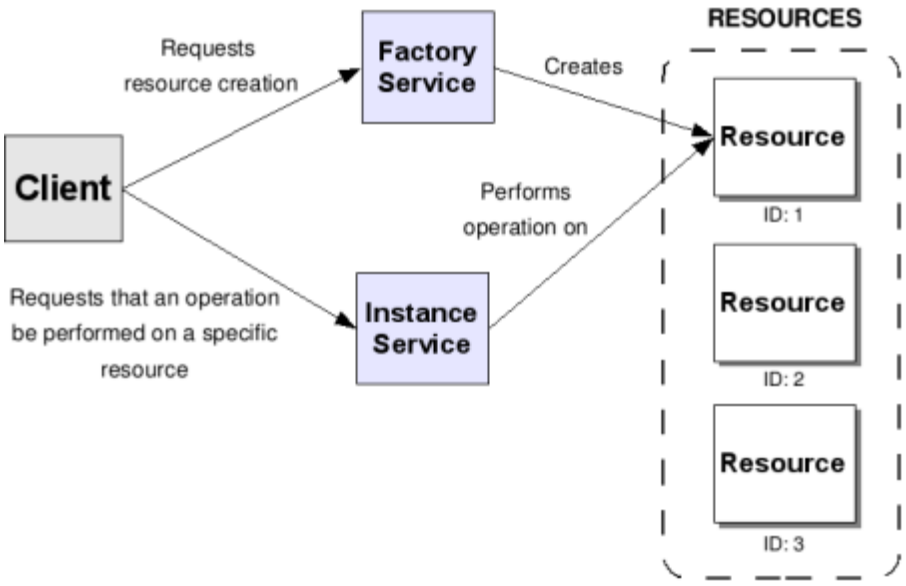


Figure 4.1, “The WS-Resource factory pattern” summarizes the relationship between these two services, the resources, and the client. Whenever the client wants to create a new resource, it does so through a factory service interface (in OO terms, a factory method). It is up to the programmer to implement the factory service. The pyGridWare infrastructure cannot create it automatically since the input parameters depend entirely on the details of the implementation. The factory service must return a reference to the created service instance and resource, in the form of an *endpoint reference*(EPR). This reference can be used to in subsequent interactions with the newly created service instance and its associated resource.

Note

Remember from ???? that endpoint references are a part of the WS-Addressing specification. EPRs allow us to uniquely address a single WS-Resource. Also, remember that a *WS-Resource* is the pairing of a service with a particular resource. In our first example client, our EPR included only the service's URI (because we had a single resource). In this chapter, our EPR will have both the service's URI and the resource's key.

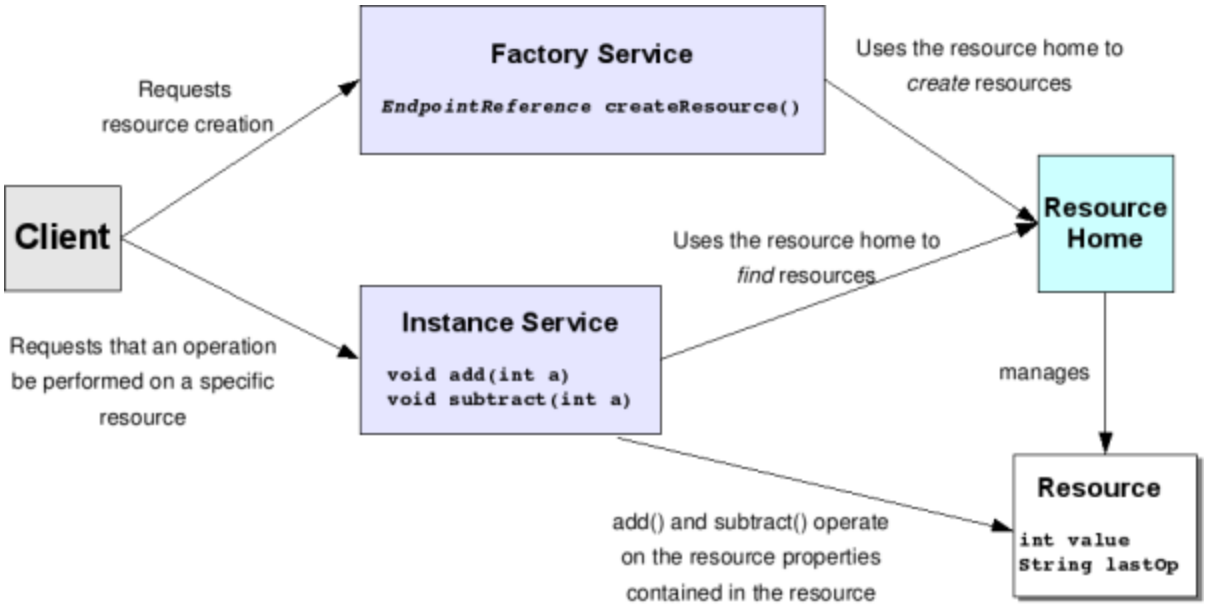
Using the EPR returned by the factory, the client can now invoke the service instance's operations. This service instance, in turn, will perform the operations using the recently created resource. If you found this confusing, you should go back and review the WS-RF concepts and terminology presented in Chapter 1.

One new term that we will use below is *resource context*, which is essentially just an aggregation of its current status.

Implementing the WS-Resource factory pattern in pyGridWare

Implementing this design pattern is very similar to the example seen in the previous chapter, with two main differences, highlighted in Figure 4.2, “Relationships between the Factory Service, the Instance Service, the Resource Home, and the Resource” (compare with ????)

Figure 4.2. Relationships between the Factory Service, the Instance Service, the Resource Home, and the Resource



- **Factory method and service instance**. To handle multiple resources we will deploy a single service that has a factory method to create service instances, this method will create new service instances and return an EPR. Within each of our service's methods we'll need to use the EPR to grab the appropriate resource (service instance), and access that resource's properites through the properties interface. This sounds complicated but it's very simple, basically we will be setting python attributes that represent the resource properties of the MathService.

- **Resource Home**. The resource home maintains all resource contexts. The factory service method will use the resource home to *create* new resources using a *NewResourceContext* method, while the instance service methods will use the resource home to find a resource using the *GetResourceContext* method.

Let's start by examining three Python modules, generated from the WSDL presented in the last section after removing the comments around the relevant create sections, containing four classes.

Download the MathService.wsdl [http://dsd.lbl.gov/gtg/projects/pyGridWare/doc/downloads/factory/MathService.wsdl], or remove the comments in the WSDL by removing all *UNCOMMENT* sections (or get it from the next section). The crucial parts of the WSDL you just uncommented are explained below.

```
<xsd:element name="Value" type="xsd:int"/>❶
<xsd:element name="LastOp" type="xsd:string"/>❷
<xsd:element name="MathResourceProperties">❸
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="tns:Value"/>
            <xsd:element ref="tns:LastOp"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>


<portType name="MathPortType"
    wsrp:ResourceProperties="tns:MathResourceProperties">❹
    ...
    ...
    <operation name="createResource">❺
            <input message="tns:CreateInputMessage"/>
            <output message="tns:CreateOutputMessage"/>
    </operation>
```

❶ *Value:* Resource Property representing the integer value of the service.

❷ *LastOp:* Resource Property representing the last operation.

❸ *MathResourceProperties:* Resource Property Declaration containing element references to the two Resource Properties.

❹ Set the wsrp:ResourceProperties attribute to the GED MathResourceProperties. Now the MathPortType is declaring its resource properties.

❺ New factory method for creating *instance* services.

Create a New Site

Use the `wsdl2web` tool to setup a site, but now provide an additional argument `--rpy`. Now the tool will provide us with a few additional pieces to make our job easier. Again ignore the warnings, but notice the last is missing since we have added a resource properties declaration to the math `portType`.

```
$ mkdir /tmp/MathSite; cd /tmp/MathSite
```

```
$ wsdl2web --rpy=MathService.rpy --script=MathFactoryClient.py
 MathService.wsdl
```

Resource Creation, Modification, and Destruction

The addition of the resource properties declaration has resulted in several changes to the generated code. We'll now examine the files where these changes have occured.

- **generated/MathService/services/MathService/MathService.py**: The generated service skeleton consists of two special methods for resource context access(*GetResourceContext*) and creation(*NewResourceContextMathPort*). The other "*wsa_*" methods are not standard WSRF operations (eg. WS-RP getResourceProperty, WS-BaseNotification Notify), thus it remains up to you to define their functionality. We'll see in the next examples how pyGridWare handles the generation of standard WSRF operations. What does this generated service skeleton provide us with for these non-standard WSRF operations? It provides us with python object representations of the request and response messages.

```
class MathServiceWSRF(MathService):

    def GetResourceContext(ps, address):
        """get a resource context"""
        return ManagerHome.getInstance().getResourceContext(ps,
 address)
    GetResourceContext = staticmethod(GetResourceContext)

    def NewResourceContextMathPort(cls, url):
        """create a new resource context"""
        ctx = MathPortTypeContext()
        ctx.setAddress(url)
        home = ctx.getResourceHome()
        key = home.create()
        home.add(ctx, key)
        ctx.setResourceID(key)
        return ctx
    NewResourceContextMathPort =
 classmethod(NewResourceContextMathPort)

    def wsa_createResource(self, ps, address, **kw):

 #http://www.globus.org/namespaces/examples/core/
MathService_instance CreateInputMessage
        request,response = MathService.wsa_createResource(self, ps,
 address)
        return request,response

    def wsa_add(self, ps, address, **kw):

 #http://www.globus.org/namespaces/examples/core/
MathService_instance AddInputMessage
        request,response = MathService.wsa_add(self, ps, address)
        return request,response
```

```
    def wsa_subtract(self, ps, address, **kw):

 #http://www.globus.org/namespaces/examples/core/
MathService_instance SubtractInputMessage
        request,response = MathService.wsa_subtract(self, ps,
 address)
        return request,response

    def wsa_getValueRP(self, ps, address, **kw):

 #http://www.globus.org/namespaces/examples/core/
MathService_instance GetValueRPInputMessage
        request,response = MathService.wsa_getValueRP(self, ps,
 address)
        return request,response
```

- **generated/MathService/resource/MathService/MathService.py**: The resource home and the resource context. A *MathPortTypeContext* instance will have an attribute `properties` that is an instance of the *MathPortType* resource properties class.

```
class ManagerHome(PersistentResourceHome, Singleton):
    def getResourceContext(self, ps, address):
        key = AddressingUtils.getResourceID(ps, Key.typecode)
        return self.get(key)


class MathPortTypeContext(ResourceContext):
    def __init__(self, properties_class=MathPortType):
        ResourceContext.__init__(self, properties_class)

    def setResourceID(self, resourceID):
        ResourceContext.setResourceID(self, Key(resourceID))
MathPortTypeContext.home = ManagerHome.getInstance()
```

- **generated/MathService/properties/MathService/MathService/MathPort.py**: The resource properties; each element of the resource property declaration is accessed and modified through a python property.

```
class MathPortType(ResourceProperties):
    declaration =
 GED("http://www.globus.org/namespaces/examples/core/
MathService_instance","MathResourceProperties")

    def __init__(self):
        ResourceProperties.__init__(self)

    def getValue(self):
        return
 self.getResourceProperty("http://www.globus.org/namespaces/
examples/core/MathService_instance","Value")
```

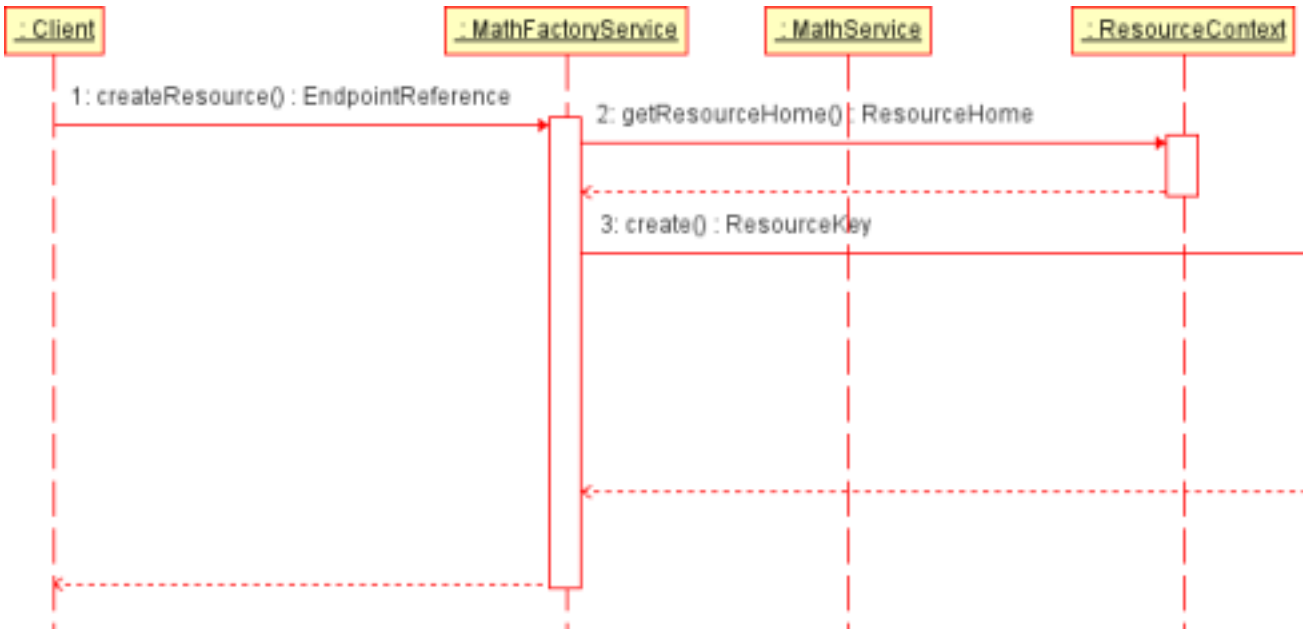
```
    def setValue(self, pyobj):
        return
 self.setResourceProperty("http://www.globus.org/namespaces/
examples/core/MathService_instance","Value", pyobj)
    Value = property(getValue, setValue, None, "RP element Value")

    def getLastOp(self):
        return
 self.getResourceProperty("http://www.globus.org/namespaces/
examples/core/MathService_instance","LastOp")
    def setLastOp(self, pyobj):
        return
 self.setResourceProperty("http://www.globus.org/namespaces/
examples/core/MathService_instance","LastOp", pyobj)
    LastOp = property(getLastOp, setLastOp, None, "RP element
 LastOp")
```

The service class **MathServiceWSRF** has two special methods.

- GetResourceContext: Uses an EPR to retrieve a resource context.
- NewResourceContextMathPort: Creates a new resource context and return it, we need to call this method in our factory method(s).

Just imagine that we actually have our our service up and ready to accept invocations from a client class, and that the service class has access to the resource home and that the resource home, in turn, has access to a bunch of resource objects. Let's start with the creation of a new resource, shown in Figure 4.3, “Sequence diagram for resource creation”.

Figure 4.3. Sequence diagram for resource creation



1. Our client only needs to know the URI of the factory service (`MathFactoryService`). With it, it can invoke the `createResource` operation. This will return an *endpoint reference*, that contains the URI and key of the recently created resource.

2. The *ResourceHome* is used to create and store *ResourceContexts*.

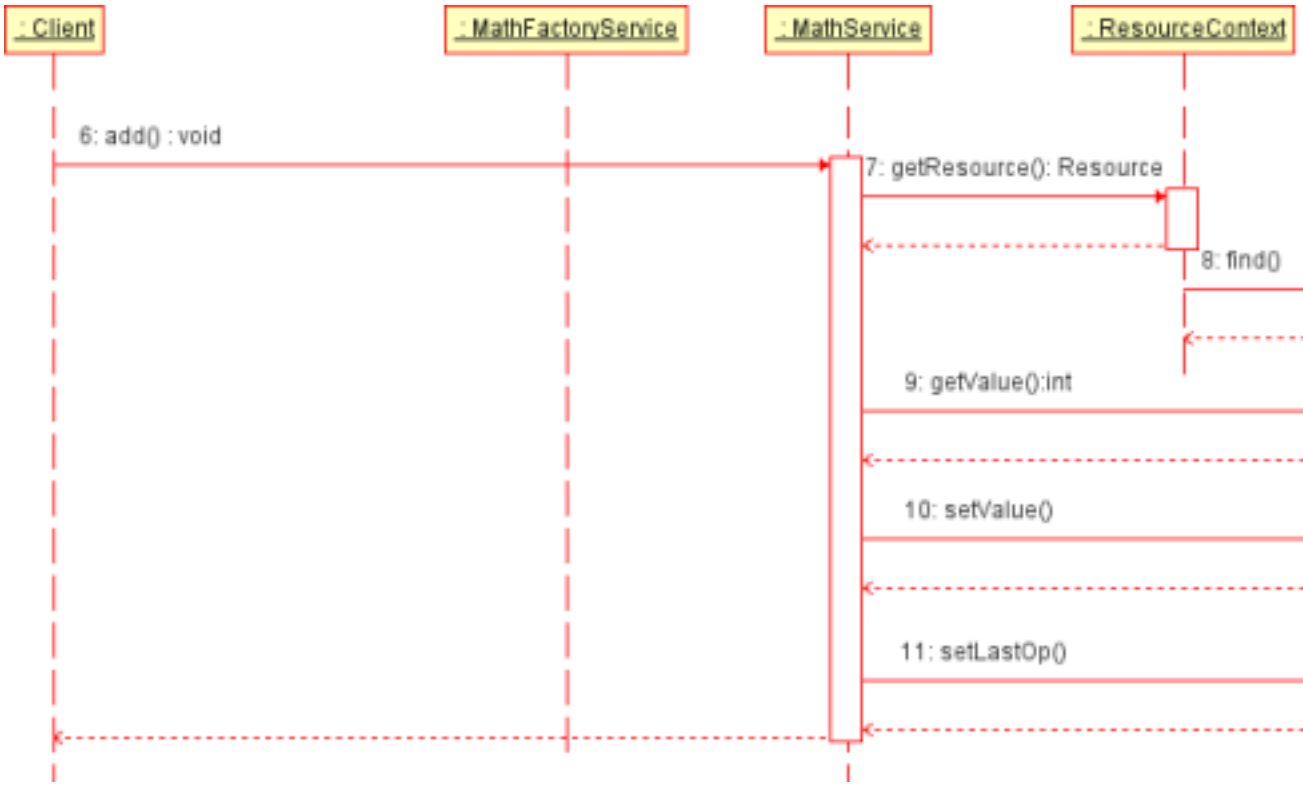
3. The factory method `createResource` has to create a new resource. This necessarily has to be done through the resource home, which is in charge of managing all the resources. However, we have to locate our resource home first. Fortunately, we don't have to deal directly with the resource home, we call `NewResourceContextMathPort` in the factory method to get a new resource.

4. Finally, the EPR is returned to the client in the response instance.

5. Skip...

Once the `createResource` call has finished, the client will have the WS-Resource's endpoint reference. In all future calls, this endpoint reference will be passed along *transparently* in all our invocations. In other words, when we call `add` or `subtract`, the service class will know what resource we're referring to. So, let's take a close look at what happens when we invoke the `add` operation, as shown in Figure 4.4, “Sequence diagram for WS-Resource invocation”.

Figure 4.4. Sequence diagram for WS-Resource invocation



6. The client invokes the `add` operation for the (`MathService`).

7. The `add` operation is stateless. First the resource context must be retrieved. The resource identifier is in the endpoint reference that is included in the address parameter. Fortunately, the generated `GetResourceContext(ps, address)` method will read the EPR and return resource context it refers to.

8. Once we have the resource context, the service can directly access state information, such as the "Value" and the "LastOp", in the resource properties "`ctx.properties`".

9. Accessing "`ctx.properties.Value`" retrieves the state.

10.Modifying "`ctx.properties.Value`" saves the state.

11.Finally, set "`ctx.properties.LastOp`" to "`ADDITION`".

Finished MathService Service

```
from generated.MathService.services.MathService.MathService import
 MathServiceWSRF

class MathService(MathServiceWSRF):

    def wsa_add(self, ps, address, **kw):
        request,response = MathServiceWSRF.wsa_add(self, ps,
 address)❶
        ctx = self.GetResourceContext(ps, address)❷
        ctx.properties.Value += request❸
        ctx.properties.LastOp = "ADDITION"❹
        return request,response

    def wsa_subtract(self, ps, address, **kw):
        request,response = MathServiceWSRF.wsa_subtract(self, ps,
 address)
        ctx = self.GetResourceContext(ps, address)
        ctx.properties.Value -= request
        ctx.properties.LastOp = "SUBTRACT"
        return request,response

    def wsa_getValueRP(self, ps, address, **kw):
        request,response = MathServiceWSRF.wsa_getValueRP(self, ps,
 address)
        ctx = self.GetResourceContext(ps, address)
        ctx.properties.LastOp = "GETVALUE"
        return request,response.__class__(ctx.properties.Value)

    #
    # The Factory Method for our Finished MathService
    #
    def wsa_createResource(self, ps, address, **kw):
        # Get request and response
        request,response = MathServiceWSRF.wsa_createResource(self,
 ps, address, **kw)

        # Create Resource Context
        from pyGridWare.utility.http import GetURLFromRequest
        ctx = self.NewResourceContextMathPort(\❻
            GetURLFromRequest(kw['request'])❻
        )

        # Initialize Value to 0
```

```
        counter = ctx.properties
        counter.Value = 0 ❼

        # Create EndPointReference and set in response
        from pyGridWare.addressing.AddressingUtils import
 AddressingUtils
        epr = AddressingUtils.createEndpointReference(ctx) ❽
        response._EndpointReference = epr ❾
        return request,response
```

❶ Call the base class to get the response and request python objects.

❷ Retrieve the resource context.

❸ Use the resource context's properties attribute to get the `Value` resource property, add the request to it, and set it to the new value.

❹ Use the resource context's properties attribute to set the `LastOp` resource property to "ADDITION".

❺ `NewResourceContextMathPort`: takes a URL and create a new resource context representing the math port.

❻ `GetURLFromRequest`: Utility function that retrieves the URL directly from the HTTP request `kw['request']`. Implication? The location of the service is abstracted from it's implementation, so the service can reply/exist at multiple locations.

❼ Initialize the resource properties of the newly created resource context.

❽ `AddressingUtils.createEndpointReference`: Utility function for creating an endpoint reference(EPR) from a resource context. The EPR is a reference to the newly created service instance.

❾ `response._EndpointReference`: Set the newly created endpoint reference in the response.

MathService.wsdl: WSDL code for a WSRF service

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="MathService"

 targetNamespace="http://www.globus.org/namespaces/examples/core/
MathService_instance"
    xmlns="http://schemas.xmlsoap.org/wsdl/"

 xmlns:tns="http://www.globus.org/namespaces/examples/core/
MathService_instance"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"

 xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties-1.2-draft-01.xsd"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<!--============================================================
```

```
                        T Y P E S

  ============================================================-->
<types>
<xsd:schema
 targetNamespace="http://www.globus.org/namespaces/examples/core/
MathService_instance"

 xmlns:tns="http://www.globus.org/namespaces/examples/core/
MathService_instance"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

        <!-- IMPORT WS-ADDRESSING -->
        <xsd:import
 namespace="http://schemas.xmlsoap.org/ws/2004/03/addressing"

 schemaLocation="http://schemas.xmlsoap.org/ws/2004/03/addressing"/>

        <!-- REQUESTS AND RESPONSES -->

        <xsd:element name="add" type="xsd:int"/>
        <xsd:element name="addResponse">
                <xsd:complexType/>
        </xsd:element>

        <xsd:element name="subtract" type="xsd:int"/>
        <xsd:element name="subtractResponse">
                <xsd:complexType/>
        </xsd:element>

        <xsd:element name="getValueRP">
                <xsd:complexType/>
        </xsd:element>
        <xsd:element name="getValueRPResponse" type="xsd:int"/>

        <xsd:element name="create">
                <xsd:complexType/>
        </xsd:element>
        <xsd:element name="createResponse">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element ref="wsa:EndpointReference"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>

        <xsd:element name="Value" type="xsd:int"/>
        <xsd:element name="LastOp" type="xsd:string"/>
        <xsd:element name="MathResourceProperties">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element ref="tns:Value"/>
                    <xsd:element ref="tns:LastOp"/>
```

```
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>

</xsd:schema>
</types>


<!--============================================================

                        M E S S A G E S

  ============================================================-->
<message name="AddInputMessage">
        <part name="parameters" element="tns:add"/>
</message>
<message name="AddOutputMessage">
        <part name="parameters" element="tns:addResponse"/>
</message>

<message name="SubtractInputMessage">
        <part name="parameters" element="tns:subtract"/>
</message>
<message name="SubtractOutputMessage">
        <part name="parameters" element="tns:subtractResponse"/>
</message>

<message name="GetValueRPInputMessage">
        <part name="parameters" element="tns:getValueRP"/>
</message>
<message name="GetValueRPOutputMessage">
        <part name="parameters" element="tns:getValueRPResponse"/>
</message>

<message name="CreateInputMessage">
        <part name="parameters" element="tns:create"/>
</message>
<message name="CreateOutputMessage">
        <part name="parameters" element="tns:createResponse"/>
</message>

<!--============================================================

                        P O R T T Y P E

  ============================================================-->

<portType name="MathPortType"
    wsrp:ResourceProperties="tns:MathResourceProperties">

        <operation name="add">
                <input message="tns:AddInputMessage"/>
                <output message="tns:AddOutputMessage"/>
        </operation>
```

```
        <operation name="subtract">
                <input message="tns:SubtractInputMessage"/>
                <output message="tns:SubtractOutputMessage"/>
        </operation>

        <operation name="getValueRP">
                <input message="tns:GetValueRPInputMessage"/>
                <output message="tns:GetValueRPOutputMessage"/>
        </operation>

        <operation name="createResource">
                <input message="tns:CreateInputMessage"/>
                <output message="tns:CreateOutputMessage"/>
        </operation>


</portType>

<!--============================================================

                        B I N D I N G

  ============================================================-->
<binding name="MathBinding" type="MathPortType">
    <soap:binding style="document"
 transport="http://schemas.xmlsoap.org/soap/http"/>

    <wsdl:operation name="add">
      <soap:operation soapAction=""/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>

    <wsdl:operation name="subtract">
      <soap:operation soapAction=""/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>

    <wsdl:operation name="getValueRP">
      <soap:operation soapAction=""/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
```

```
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>

    <wsdl:operation name="createResource">
      <soap:operation soapAction=""/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
</binding>

<!--============================================================

                      S E R V I C E

  ============================================================-->
<service name="MathService">
    <port name="MathPort" binding="MathBinding">
       <soap:address location="http://localhost:8080/wsrf/services/"/>
    </port>
</service>

</definitions>
```

Installing the MathService.rpy script

The finished MathService can be installed directly as an "*rpy*" script in the "*services* sub-directory where the service container is executed.

```
%ls services/
MathService.rpy
%./start-container.sh -l server.log
```

The service container above appears to have only a MathService, but other services that are transparent to the user such as SubscriptionManager and NotificationProducer are configured in the `server-config.tac` file.

Note

Finished MathService.rpy [http://dsd.lbl.gov/gtg/projects/pyGridWare/doc/downloads/factory/MathService.rpy] script

```
from generated.MathService.services.MathService.MathService import
 MathServiceWSRF

class Service(MathServiceWSRF):
```

```
    def wsa_add(self, ps, address, **kw):
        request,response = MathServiceWSRF.wsa_add(self, ps, address)
        ctx = self.GetResourceContext(ps, address)
        ctx.properties.Value += request
        ctx.properties.LastOp = "ADDITION"
        return request,response

    def wsa_subtract(self, ps, address, **kw):
        request,response = MathServiceWSRF.wsa_subtract(self, ps,
address)
        ctx = self.GetResourceContext(ps, address)
        ctx.properties.Value -= request
        ctx.properties.LastOp = "SUBTRACT"
        return request,response

    def wsa_getValueRP(self, ps, address, **kw):
        request,response = MathServiceWSRF.wsa_getValueRP(self, ps,
address)
        ctx = self.GetResourceContext(ps, address)
        ctx.properties.LastOp = "GETVALUE"
        return request,response.__class__(ctx.properties.Value)

    #
    # The Factory Method for our Finished MathService
    #
    def wsa_createResource(self, ps, address, **kw):
        # Get request and response
        request,response = MathServiceWSRF.wsa_createResource(self,
ps, address, **kw)

        # Create Resource Context
        from pyGridWare.utility.http import GetURLFromRequest
        ctx = self.NewResourceContextMathPort(\
            GetURLFromRequest(kw['request'])
        )

        # Initialize Value to 0
        counter = ctx.properties
        counter.Value = 0

        # Create EndPointReference and set in response
        from pyGridWare.addressing.AddressingUtils import
AddressingUtils
        epr = AddressingUtils.createEndpointReference(ctx)
        response._EndpointReference = epr
        return request,response
(1)
resource = Service()
```

(1) extra line to create the `resource` so twisted will recognize it as such.

The ResourceHome and Persistence

Note

Sorry, this section needs to be redone.

Sample Client using an *EndpointReference*

We will try out our service first with a simple client that creates a new resource and performs a couple operations on it. Starting from the client that was generated by `wsdl2web` all that's needed are a few rpc calls to demonstrate WSRF resource's behavior. Download MathFactoryClient.py [http://dsd.lbl.gov/gtg/projects/pyGridWare/doc/downloads/factory/MathFactoryClient.py]

```
#!/usr/bin/env python
############################################################################
# MathFactoryClient.py
# See LBNLCopyright for copyright notice!
############################################################################
from twisted.python import log
from twisted.internet import reactor

import ZSI
from pyGridWare.utility.scripts.client import GetBasicOptParser,
 GetPortKWArgs, SetUp
from generated.MathService.stubs import MathService as CLIENT


def main(**kw):
    locator = CLIENT.MathServiceLocator()
    port = locator.getMathPort(**kw) ❶

    msg = port.createResource(CLIENT.CreateInputMessage()) ❷
    iport =
 locator.getMathPort(endPointReference=msg.EndpointReference, **kw) ❸
    for i in range(10):
        iport.add(CLIENT.AddInputMessage(i)) ❹

    msg = iport.subtract(CLIENT.SubtractInputMessage(10))

    msg = iport.getValueRP(CLIENT.GetValueRPInputMessage())

    # Factory METHOD Just guessing here
    #response = port.create(CLIENT.CreateRequest())
    #kw['endPointReference'] = response._EndpointReference
    #iport = locator.getMathPort(**kw)
    reactor.stop()
    print "MSG: ", msg
    print "CORRECT: ", msg == sum(range(10)) - 10
```

```
if __name__ == '__main__':
    op = GetBasicOptParser()
    (options, args) = op.parse_args()
    SetUp(options)
    kw = GetPortKWArgs(options)
    reactor.callWhenRunning(main, **kw)
    reactor.run()
```

1. Here we obtain a reference to a `MathPortType` instance.
2. Once we have the MathPortType, we use it to invoke the `createResource` operation. This operation returns an endpoint reference, inside a `CreateResourceResponse` object. This endpoint reference includes both the instance service's URI *and* the new resource's identifier. In the next client we will take a peek inside the endpoint reference.
3. Using the instance EPR, we obtain a new `MathPortType` which will now refer to the instance service.
4. We now use the `MathPortType` to invoke `add`, `subtract`, and `getValueRP`.

run the client:

```
./MathFactoryClient.py -u
 http://127.0.0.1:9080/wsrf/services/MathService -d 0
```

If all goes well, you should see the following:

```
MSG:  35
CORRECT:  True
```

If you run it again, you will get the exact same result. This is because we are creating a new resource every time we run the client.

```
MSG:  35
CORRECT:  True
```

Chapter 5. Resource Properties

In the previous chapters we have seen how state information in the service is stored inside a resource and, more specifically, in *resource properties*. However, our interaction with resource properties was very limited: our service could modify their values, and we could only access one particular resource property (`Value`) using the `GetValueRP` operation. In this chapter we will see all the some of the tools that will allow us to work with resource properties.

A closer look at resource properties

Before we begin, we need to take a closer look at how resource properties are represented and handled internally in our service. First of all, let's recall how our resource properties are declared in all the examples we've seen so far:

```
<types>
<xsd:schema
 targetNamespace="http://www.globus.org/namespaces/examples/core/
MathService_instance"

 xmlns:tns="http://www.globus.org/namespaces/examples/core/
MathService_instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">


 <!-- REQUESTS AND RESPONSES -->

 <!-- ... -->



 <!-- RESOURCE PROPERTIES -->

 <xsd:element name="Value" type="xsd:int"/>
 <xsd:element name="LastOp" type="xsd:string"/>

 <xsd:element name="MathResourceProperties">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element ref="tns:Value" minOccurs="1" maxOccurs="1"/>
   <xsd:element ref="tns:LastOp" minOccurs="1" maxOccurs="1"/>
  </xsd:sequence>
 </xsd:complexType>
 </xsd:element>

</xsd:schema>
</types>
```

Notice how we're using XML Schema to declare an element named `MathResourceProperties` that must contain a `Value` element and a `LastOp` element. The `Value` element, in turn, is declared to contain an integer (`xsd:int`) and the `LastOp` element, a string (`xsd:string`).

In the previous examples, we have simply interpreted this as meaning "Our service has two resource properties, `Value` of type integer and `LastOp` of type string". The generated resource class has python attributes representing each of the resource properties.

Resource properties are declared in XML Schema in order to be exchanged with other entities (clients, other services, etc.) as an XML document. This XML representation is called the *resource property document*. For example, the following is an example of how our service's RP document might look like at a given point:

```
<MathResourceProperties
 xmlns:tns="http://www.globus.org/namespaces/examples/core/
MathService_instance">
 <tns:Value>50</tns:Value>
 <tns:LastOp>ADDITION</tns:LastOp>
</MathResourceProperties>
```

It is important to be familiar with this representation because many operations related with resource properties are better explained in terms of how that operation modifies the RP document. For example, let's suppose our resource properties are declared the following way:

```
<!-- RESOURCE PROPERTIES -->

<xsd:element name="Value" type="xsd:int"/>
<xsd:element name="LastOp" type="xsd:string"/>

<xsd:element name="MathResourceProperties">
<xsd:complexType>
 <xsd:sequence>
  <xsd:element ref="tns:Value" minOccurs="1" maxOccurs="unbounded"/>
  <xsd:element ref="tns:LastOp" minOccurs="1" maxOccurs="unbounded"/>
 </xsd:sequence>
</xsd:complexType>
</xsd:element>
```

Notice how we are allowing `Value` and `LastOp` to occur at least one time, with no other limit (`unbounded`). Although internally this will be implemented as an array of integers and an array of strings, the RP document could look like this at a given point:

```
<MathResourceProperties
 xmlns:tns="http://www.globus.org/namespaces/examples/core/
MathService_instance">
 <tns:Value>10</tns:Value>
 <tns:Value>30</tns:Value>
 <tns:Value>50</tns:Value>
 <tns:Value>40</tns:Value>
 <tns:LastOp>ADDITION</tns:LastOp>
 <tns:LastOp>ADDITION</tns:LastOp>
 <tns:LastOp>ADDITION</tns:LastOp>
 <tns:LastOp>SUBTRACTION</tns:LastOp>
</MathResourceProperties>
```

Later on, for example, we will talk about "inserting a new resource property `LastOp` with value `ADDITION`". This doesn't mean that we are *declaring* a new RP but, rather, that we are inserting a new

element `LastOp` inside our resource property document. Again, it is useful to be aware of how resource properties are represented in XML.

Standard interfaces

An entire WSRF specification, WS-ResourceProperties, is devoted to RPs, RP documents, and to a set of standard portTypes we can use to interact with a service's RPs. Each of these four portTypes exposes a single operation, with the same name as the portType. In this chapter's example we will use all of these portTypes.

GetResourceProperty

This portType allows us to access the value of any resource property given its QName. This portType provides a general way of accessing RPs without the need of an individual `get` operation for each RP (recall that, in previous chapters, we used the `GetValueRP` operation to access the `Value` resource property).

GetMultipleResourceProperties

This portType allows us to access the value of several resource properties at once, given each of their QNames.

SetResourceProperties

This portType allows us to request one or several modifications on a service's RPs. In particular we can perform the following operations:

- Update: Change the value of a RP with a new value.
- Insert: Add a new RP with a given value.
- Delete: Eliminate all occurrences of a certain RP.

Again, note that the `SetResourceProperties` portType has a single operation (not three separate ones). We will use the parameters of the `SetResourceProperties` to specify what action (update, insert, or delete) we want to carry out.

QueryResourceProperties

This portType allows us to perform complex queries on the RP document. Currently, the query language used is XPath.

Extending The MathPortType: A New WSDL file

We will now write and deploy a new service that exposes all the WS-ResourceProperty portTypes through compositional inheritance. What is compositional inheritance? Basically copying & pasting operations from the WSRP portTypes to the MathPortType. Our client application will, in turn, make a call to some of these portTypes.

In this chapter we must use a new WSDL file because we want to extend from new portTypes, which necessarily changes our service's interface. In this WSDL the "`getValueRP`" operation has been removed. The GetResourceProperty, SetResourceProperty, GetMultipleResourceProperties, and

QueryResourceProperties operations were copied from their respective WSRP portTypes into the MathPortType. In theMathBinding we also need to define each new operation.

Note

Download: MathRPService.wsdl [http://dsd.lbl.gov/gtg/projects/pyGridWare/doc/downloads/rp/MathRPService.wsdl]

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="MathServiceRP"

 targetNamespace="http://www.globus.org/namespaces/examples/core/
MathService_instance_rp" ❶
    xmlns="http://schemas.xmlsoap.org/wsdl/"

 xmlns:tns="http://www.globus.org/namespaces/examples/core/
MathService_instance_rp"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"

 xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties-1.2-draft-01.xsd"

 xmlns:wsrpw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties-1.2-draft-01.wsdl" ❷
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">


<wsdl:import ❸
    namespace=

 "http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-
1.2-draft-01.wsdl"
    location="wsrf/properties/WS-ResourceProperties.wsdl" />


<!--============================================================

                     T Y P E S

  ============================================================-->
<types>
<xsd:schema
 targetNamespace="http://www.globus.org/namespaces/examples/core/
MathService_instance_rp"

 xmlns:tns="http://www.globus.org/namespaces/examples/core/
MathService_instance_rp"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

        <!-- IMPORT WS-ADDRESSING -->
```

```
        <xsd:import
 namespace="http://schemas.xmlsoap.org/ws/2004/03/addressing"

 schemaLocation="http://schemas.xmlsoap.org/ws/2004/03/addressing"/>

        <!-- REQUESTS AND RESPONSES -->

        <xsd:element name="add" type="xsd:int"/>
        <xsd:element name="addResponse">
                <xsd:complexType/>
        </xsd:element>
        <xsd:element name="subtract" type="xsd:int"/>
        <xsd:element name="subtractResponse">
                <xsd:complexType/>
        </xsd:element>

        <xsd:element name="create">
                <xsd:complexType/>
        </xsd:element>
        <xsd:element name="createResponse">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element ref="wsa:EndpointReference"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>

        <!-- MathService Resource Properties -->

        <xsd:element name="Value" type="xsd:int"/>
        <xsd:element name="LastOp" type="xsd:string"/>
        <xsd:element name="MathResourceProperties">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element ref="tns:Value"/>
                    <xsd:element ref="tns:LastOp"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>

</xsd:schema>
</types>

<!--============================================================

                    M E S S A G E S

  ============================================================-->
<message name="AddInputMessage">
        <part name="parameters" element="tns:add"/>
</message>
<message name="AddOutputMessage">
        <part name="parameters" element="tns:addResponse"/>
</message>
```

```
<message name="SubtractInputMessage">
        <part name="parameters" element="tns:subtract"/>
</message>
<message name="SubtractOutputMessage">
        <part name="parameters" element="tns:subtractResponse"/>
</message>

<message name="CreateInputMessage">
        <part name="parameters" element="tns:create"/>
</message>
<message name="CreateOutputMessage">
        <part name="parameters" element="tns:createResponse"/>
</message>


<!--============================================================

                      P O R T T Y P E

  ============================================================-->

<portType name="MathPortType"
    wsrp:ResourceProperties="tns:MathResourceProperties">


 <operation name="add"> ❹
  <input message="tns:AddInputMessage"/>
  <output message="tns:AddOutputMessage"/>
 </operation>

 <operation name="subtract">
  <input message="tns:SubtractInputMessage"/>
  <output message="tns:SubtractOutputMessage"/>
 </operation>

 <operation name="createResource">
  <input message="tns:CreateInputMessage"/>
  <output message="tns:CreateOutputMessage"/>
 </operation>


 <operation name="GetResourceProperty"> ❺
  <input name="GetResourcePropertyRequest"
                   message="wsrpw:GetResourcePropertyRequest"

 wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties/GetResourceProperty"/>
                <output name="GetResourcePropertyResponse"
                    message="wsrpw:GetResourcePropertyResponse"

 wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties/GetResourcePropertyResponse"/>
                <fault name="InvalidResourcePropertyQNameFault"
```

```
                    message="wsrpw:InvalidResourcePropertyQNameFault"/
>
                <fault name="ResourceUnknownFault"
                    message="wsrpw:ResourceUnknownFault"/>
 </operation>

    <operation name="GetMultipleResourceProperties"> ❻
      <input name="GetMultipleResourcePropertiesRequest"
 message="wsrpw:GetMultipleResourcePropertiesRequest"
 wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties/GetMultipleResourceProperties"/>
      <output name="GetMultipleResourcePropertiesResponse"
 message="wsrpw:GetMultipleResourcePropertiesResponse"
 wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties/GetMultipleResourcePropertiesResponse"/>
      <fault name="InvalidResourcePropertyQNameFault"
 message="wsrpw:InvalidResourcePropertyQNameFault"/>
      <fault name="ResourceUnknownFault"
 message="wsrpw:ResourceUnknownFault"/>
    </operation>

    <operation name="SetResourceProperties"> ❼
      <input name="SetResourcePropertiesRequest"
 message="wsrpw:SetResourcePropertiesRequest" wsa:Action="http://
docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties/
SetResourceProperties"/>
      <output name="SetResourcePropertiesResponse"
 message="wsrpw:SetResourcePropertiesResponse" wsa:Action="http://
docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties/
SetResourcePropertiesResponse"/>
      <fault name="InvalidResourcePropertyQNameFault"
 message="wsrpw:InvalidResourcePropertyQNameFault"/>
      <fault name="UnableToModifyResourcePropertyFault"
 message="wsrpw:UnableToModifyResourcePropertyFault"/>
      <fault name="SetResourcePropertyRequestFailedFault"
 message="wsrpw:SetResourcePropertyRequestFailedFault"/>
      <fault name="ResourceUnknownFault"
 message="wsrpw:ResourceUnknownFault"/>
      <fault name="InvalidSetResourcePropertiesRequestContentFault"
 message="wsrpw:InvalidSetResourcePropertiesRequestContentFault"/>
    </operation>

    <operation name="QueryResourceProperties"> ❽
      <input name="QueryResourcePropertiesRequest"
 message="wsrpw:QueryResourcePropertiesRequest" wsa:Action="http://
docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties/
QueryResourceProperties"/>
      <output name="QueryResourcePropertiesResponse"
 message="wsrpw:QueryResourcePropertiesResponse" wsa:Action="http://
docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties/
QueryResourcePropertiesResponse"/>
```

```
      <fault name="UnknownQueryExpressionDialectFault"
 message="wsrpw:UnknownQueryExpressionDialectFault"/>
      <fault name="InvalidResourcePropertyQNameFault"
 message="wsrpw:InvalidResourcePropertyQNameFault"/>
      <fault name="QueryEvaluationErrorFault"
 message="wsrpw:QueryEvaluationErrorFault"/>
      <fault name="InvalidQueryExpressionFault"
 message="wsrpw:InvalidQueryExpressionFault"/>
      <fault name="ResourceUnknownFault"
 message="wsrpw:ResourceUnknownFault"/>
    </operation>

</portType>

<!--================================================================

                        B I N D I N G

  ================================================================-->
<binding name="MathBinding" type="MathPortType"> (9)
    <soap:binding style="document"
 transport="http://schemas.xmlsoap.org/soap/http"/>

    <wsdl:operation name="add">
      <soap:operation soapAction=""/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>

    <wsdl:operation name="subtract">
      <soap:operation soapAction=""/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>

    <wsdl:operation name="createResource">
      <soap:operation soapAction=""/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
```

```
    <wsdl:operation name="GetResourceProperty">
      <soap:operation soapAction=""/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>

    <wsdl:operation name="GetMultipleResourceProperties">
      <soap:operation soapAction=""/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>

    <wsdl:operation name="SetResourceProperties">
      <soap:operation soapAction=""/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>

    <wsdl:operation name="QueryResourceProperties">
      <soap:operation soapAction=""/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
</binding>

<!--============================================================

                      S E R V I C E

  ============================================================-->
<service name="MathService">
    <port name="MathPort" binding="MathBinding">
       <soap:address location="http://localhost:8080/wsrf/services/"/>
    </port>
</service>

</definitions>
```

Note

Copy & Paste WSDL definition to "*/tmp/MathRPService.wsdl*".

1. Notice how we declare a new target namespace for our new WSDL interface.
2. This namespace declaration (`wsrpw`) was already present in previous examples. We need to declare the WS-ResourceProperties namespace if we want to use the messages associated with the WSRP portTypes defined in that specification.
3. We must import the WS-ResourceProperties WSDL file, where the portTypes and associated messages are actually defined. Futhermore this WSDL file must exist at the *location* specified in the import.
4. Our example exposes the `add`, `subtract`, and `createResource` operations in addition to the WS-ResourceProperty portType operations.
5. We've eliminated the `GetValueRP` operation (although not shown above, the WSDL file also lacks the corresponding `GetValueRP` messages and elements). Since we are now going to use the `GetResourceProperty` portType, there is no need to expose an explicit `GetValueRP` operation.
6. GetMultipleResourceProperties portType operation.
7. SetResourceProperties portType operation.
8. QueryResourceProperties portType operation.
9. We need to add each additional operation to the MathBinding, this can be accomplished by simply copying one of the other operations and changing the name attribute.

Create a New Site

```
% mkdir /tmp/MathRPSite
% cd /tmp/MathRPSite
% mv /tmp/MathRPService.wsdl .
% wsdl2web --script=client.py --rpy=MathService.rpy MathRPService.wsdl

% wsdl2web --script=client.py --rpy=MathService.rpy MathRPService.wsdl
Failure:  [Errno 2] No such file or directory:
 'wsrf/properties/WS-ResourceProperties.wsdl'

% cp -r ~/Desktop/Workspace/Python/pyGridWare/share/schema/wsrf .

% wsdl2web --script=client.py --rpy=MathService.rpy MathRPService.wsdl
Failure:  [Errno 2] No such file or directory: 'ws/xml.xsd'

% cp -r ~/Desktop/Workspace/Python/pyGridWare/share/schema/ws .

% wsdl2web --script=client.py --rpy=MathService.rpy MathRPService.wsdl
ZSI/wstools/XMLSchema.py:1214: UserWarning: annotation is ignored
```

In the next two sections I'll present new rpy and client scripts that you should copy & paste over the auto generated *services/MathService.rpy* and *client.py*.

New MathService.rpy Script

We no longer need to implement the `getValueRP` operation. The WS-ResourceProperties portTypes, or operations, are generated in the `MathServiceWSRF` class. But we need to implement the add, subtract and createResource methods.

Note

Download: MathService.rpy [http://dsd.lbl.gov/gtg/projects/pyGridWare/doc/downloads/rp/MathService.rpy]

```
############################################################################
# Automatically generated by wsdl2web.py
# See LBNLCopyright for copyright notice!
############################################################################
from generated.MathRPService.services.MathRPService.MathService import
 MathServiceWSRF

class Service(MathServiceWSRF):

    def wsa_add(self, ps, address, **kw):
        request,response = MathServiceWSRF.wsa_add(self, ps, address)
        ctx = self.GetResourceContext(ps, address)
        ctx.properties.Value += request
        ctx.properties.LastOp = "ADDITION"
        return request,response


    def wsa_subtract(self, ps, address, **kw):
        request,response = MathServiceWSRF.wsa_subtract(self, ps,
 address)
        ctx = self.GetResourceContext(ps, address)
        ctx.properties.Value -= request
        ctx.properties.LastOp = "SUBTRACT"
        return request,response

    #
    # The Factory Method for our Finished MathService
    #
    def wsa_createResource(self, ps, address, **kw):
        # Get request and response
        request,response = MathServiceWSRF.wsa_createResource(self,
 ps, address, **kw)

        # Create Resource Context
        from pyGridWare.utility.http import GetURLFromRequest
        ctx = self.NewResourceContextMathPort(\
            GetURLFromRequest(kw['request'])
        )

        # Initialize Value to 0
        counter = ctx.properties
```

```
        counter.Value = 0

        # Create EndPointReference and set in response
        from pyGridWare.addressing.AddressingUtils import
 AddressingUtils
        epr = AddressingUtils.createEndpointReference(ctx)
        response.EndpointReference = epr
        return request,response

resource = Service()
```

Client code

Our client application will make calls to some of the WS-ResourceProperties portTypes. The next example will use more complex resource properties and then we will see how to invoke the rest of the portTypes.

In the examples that follow instances of python stub classes are directly manipulated and access through attributes. The python attributes of an object represent XML Schema element declarations, there is a predictable mapping. Above the tests are a few relevant XML Schema samples.

Note

Download Complete Client: client.py [http://dsd.lbl.gov/gtg/projects/pyGridWare/doc/downloads/rp/client.py]

```
#!/usr/bin/env python
############################################################################
# Automatically generated by wsdl2web.py
# See LBNLCopyright for copyright notice!
############################################################################
from twisted.python import log
from twisted.internet import reactor

import ZSI
from pyGridWare.utility.scripts.client import GetBasicOptParser,
 GetPortKWArgs, SetUp
from generated.MathRPService.stubs import MathRPService as CLIENT


def GetResourceProperty(iport):
    """ Replace GetResourceProperty """

def SetResourceProperty(iport):
    """ Replace SetResourceProperty """

def GetMultipleResourceProperties(iport):
    """ Replace GetMultipleResourceProperties """


def main(*argv, **kw):
    locator = CLIENT.MathServiceLocator()
    port = locator.getMathPort(**kw)
```

```
    msg = port.createResource(CLIENT.CreateInputMessage())
    iport =
 locator.getMathPort(endPointReference=msg.EndpointReference, **kw)

    if len(argv) > 0:
        method = eval(argv[0])
        method(iport)

    reactor.stop()


if __name__ == '__main__':
    op = GetBasicOptParser()
    (options, args) = op.parse_args()
    SetUp(options)
    kw = GetPortKWArgs(options)
    reactor.callWhenRunning(main, *args, **kw)
    reactor.run()
```

Note

Copy & Paste to client.py, we'll add the missing sections below.

Invoking `GetResourceProperty`

XML Schema GetResourceProperty global element declaration(GED), its type is QName. XML Schema GetResourcePropertyResponse GED. It contains an element wildcard *any*, and the value of its *maxOccurs* facet is unbounded.

```
<!-- ========== Message Types for GetResourceProperty  ============ --
>

       <xsd:element name="GetResourceProperty"
                    type="xsd:QName" />

       <xsd:element name="GetResourcePropertyResponse" >
         <xsd:complexType>
           <xsd:sequence>
             <xsd:any  minOccurs="0" maxOccurs="unbounded" />
           </xsd:sequence>
         </xsd:complexType>
       </xsd:element>



def GetResourceProperty(iport):
    for i in range(10):
        iport.add(CLIENT.AddInputMessage(i))

    iport.subtract(CLIENT.SubtractInputMessage(10))

    msg = iport.GetResourceProperty(\
```

```
        CLIENT.GetResourcePropertyRequest(\

 ("http://www.globus.org/namespaces/examples/core/
MathService_instance_rp","Value"),
        )
    )


    print "MSG: ", msg ❷
    print "PROPERTIES: ", msg.Any
    print "PROPERTY: ", msg.Any[0] ❸
    print "CORRECT: ", msg.Any[0] == sum(range(10)) - 10


% ./client.py -u http://127.0.0.1:9080/wsrf/services/MathService -d 0
 GetResourceProperty
MSG:
 <pyGridWare.generated.types.resourceproperties.GetResourcePropertyResponse_Holder
 instance at 0x13480a8>
PROPERTIES:  [35]
PROPERTY:  35
CORRECT:  True
```

❶ We first invoke the `GetResourceProperty` operation on our portType. Take into account that, since our `MathPortType` portType *extends* from the standard `GetResourceProperty` portType, our portType also includes a `GetResourceProperty` operation. The only parameter we have to include is the QName, which is specified as a namespace name tuple, of the RP we want to retrieve.

❷ The return value is of type `GetResourcePropertyResponse`, a python stub class.

❸ We can access the RPs' values from the `GetResourcePropertyResponse` python object. The `GetResourcePropertyResponse's` any wildcard element will contain zero, one, or many RPs. The RP values are set at the `_any` python attribute name, but it's easier to use the corresponding `Any` python property which works just like an attribute. Since the any element is repeatable (*maxOccurs="unbounded"*), the attribute's value will be a list.

Invoking `SetResourceProperties` to update

XML Schema Type Definiton and GED for Update and SetResourceProperties.

```
<xsd:complexType name="UpdateType">
  <xsd:sequence>
    <xsd:any processContents="lax"
             minOccurs="1" maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>
<xsd:element name="Update"
             type="wsrp:UpdateType"/>

<xsd:element name="SetResourceProperties">
  <xsd:complexType>
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element ref="wsrp:Insert"/>
```

```
            <xsd:element ref="wsrp:Update"/>
            <xsd:element ref="wsrp:Delete"/>
          </xsd:choice>
        </xsd:complexType>
       </xsd:element>

      <xsd:element name="SetResourcePropertiesResponse" >
         <xsd:complexType />
      </xsd:element>
```

Note

Replace test function and add a SetResourceProperties call

```
def SetResourceProperties(iport):
    iport.add(CLIENT.AddInputMessage(111))

    # Take the Value ResourceProperty and create a new instance
    from
 generated.MathRPService.properties.MathService.MathRPService.MathPort
 import MathPortType
    properties = MathPortType() ❶
    properties.Value = 100 ❷
    req = CLIENT.SetResourcePropertiesRequest() ❸

    req.Update = [req.new_Update(),] ❹
    req.Update[0].Any = [properties.Value,] ❺

    smsg = iport.SetResourceProperties(req) ❻

    assert
 ("http://www.globus.org/namespaces/examples/core/
MathService_instance_rp","Value") ==\
        (properties.Value.typecode.nspname,
 properties.Value.typecode.pname),\
        "Grab Value ResourceProperty GED's namespace, name."

    msg = iport.GetResourceProperty(\
        CLIENT.GetResourcePropertyRequest(\
            (properties.Value.typecode.nspname,
 properties.Value.typecode.pname),
        )
    )

    print "SetResourceProperties MSG: ", smsg
    print "PROPERTIES: ", msg.Any
    print "CORRECT: ", msg.Any[0] == 100
```

```
%./client.py -u http://127.0.0.1:9080/wsrf/services/MathService -d 0
 SetResourceProperties
SetResourceProperties MSG:
 <pyGridWare.generated.types.resourceproperties.SetResourcePropertiesResponse_Hold
 instance at 0x107d080>
PROPERTIES:  [100]
CORRECT:  True
```

1. Create an instance of the ResourceProperties we would like to remotely access.
2. Set the new `Value`. This wraps the integer in a serializable object.
3. Create a `SetResourcePropertiesRequest` object which will represents the `SetResourcePropertiesRequest` message specified in the WSDL. This object can contain insert, update, and delete actions.
4. Create a python object and set it in the `SetResourceProperties` instance's `_Update` python attribute to specify an update. TODO: Use new metaclass for pyclass to make this transparent once I'm convinced it's stable.
5. Add the Value RP to the list of RP's to be updated.
6. Finally, we invoke `SetResourceProperties`.

Invoking `GetMultipleResourceProperties`

Note

Replace test function and add a GetMultipleResourceProperties call

```
def GetMultipleResourceProperties(iport):
    for i in range(10):
        iport.add(CLIENT.AddInputMessage(i))

    req = CLIENT.GetMultipleResourcePropertiesRequest() ❶

    req.ResourceProperty =
 [("http://www.globus.org/namespaces/examples/core/
MathService_instance_rp","Value"),

 ("http://www.globus.org/namespaces/examples/core/
MathService_instance_rp","LastOp"),]❷

    msg = iport.GetMultipleResourceProperties(req) ❸

    print "MSG: ", msg
    for any in msg.Any: ❹
        print "Resource Property(%s) = " %any.__class__,any
```

```
% ./client.py -u http://127.0.0.1:9080/wsrf/services/MathService -d 0
 GetMultipleResourceProperties
MSG:
 <pyGridWare.generated.types.resourceproperties.GetMultipleResourcePropertiesRespo
 instance at 0x134a468>
```

```
Resource Property(<class
 'pyGridWare.generated.types.mathservicerp._Value_immutable_holder'>)
 =  45
Resource Property(<class
 'pyGridWare.generated.types.mathservicerp._LastOp_immutable_holder'>)
 =  ADDITION
```

1. First, we need to create a `GetMultipleResourcePropertiesRequest` object that represents the WSDL input message for the GetMultipleResourceProperties operation.
2. The local element `ResourceProperty` is a repreatable QName type. Thus to specify the RPs `Value` and `LastOp`, we need to append *(namespace, name)* tuples representing these GEDs. The underlying architecture expects these tuples to be in a list because the `maxOccurs` facet is greater than one.
3. Next, we invoke the `GetMultipleResourceProperties`. The return value is of a `GetMultipleResourcePropertiesResponse_Holder` instance, which represents the WSDL output message for the GetMultipleResourceProperties operation.
4. The RPs returned are accessed through the `Any` python property (it wraps the `_any` python attribute), which is a list representing the `<any>` wildcard element (WS-ResourceProperties [http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.wsdl]). In this example we request and received two RPs. The RPs are instances of `class _Value_immutable_holder` and `_LastOp_immutable_holder`.

Chapter 6. Lifecycle Management

In this chapter we will see the two lifecycle management solutions offered by the WS-ResourceLifetime specification. Since lifecycle management mainly makes sense when we have several resources, the examples will focus on explaining what modifications are necessary to the example seen in ???? (the factory/instance example).

Immediate destruction

Immediate destruction is the simplest type of lifecycle management. It allows us to request that a resource be destroyed immediately by invoking a `destroy` operation in the *instance* service. Notice how, even though the *factory* service is responsible for creating the resources, destruction must be requested to each individual resource through the instance service.

Overview: Directions to add immediate destruction to our service

Note

The WSDL presented in the last section of this chapter describes a service with WS-ResourceLifetime properties. You can skip most of this section and just generate and install the math service bindings. In this section I present a high level explanation of how to add immediate destruction to the math service.

To add immediate destruction to our service, we need to add the standard WSRF `ImmediateResourceTermination` portType to our service through compositional inheritance. This portType adds a `Destroy` operation to our portType that will instruct the current resource to terminate itself immediately. In sum, we need to copy the WSRF `ImmediateResourceTermination` portType `Destroy` operation to the MathPortType and add a `Destroy` binding operation to MathBinding.

Note

Download the MathService [http://dsd.lbl.gov/gtg/projects/pyGridWare/doc/downloads/lifetime/MathLTService.wsdl] WSDL, it is also at the end of this chapter. Then grab the "*MathService.rpy*" from the last chapter. The WS-ResourceLifetime operations are implemented for us. This RPY script will work with one modification to an `import` statement since the directory structure of our generated code is a little different. I'll accomplish this using sed.

```
% cd /tmp/MathLTSite
% curl -O
 http://dsd.lbl.gov/gtg/projects/pyGridWare/doc/downloads/lifetime/
MathLTService.wsdl
% wsdl2web --script=client.py --rpy=MathService.rpy MathLTService.wsdl
  ...
  ...
% grep MathLTService services/MathService.rpy
from generated.MathLTService.services.MathLTService.MathService import
 MathServiceWSRF
```

```
% cat /tmp/MathRPSite/services/MathService.rpy | sed
 s/MathRPService/MathLTService/g > services/MathService.rpy
```

Generated WSRF Service: `generated/MathLTService/services/MathLTService/MathService.py`

Note

This class has been generated for you. The WSRF operations specified in the WSDL definition are implemented in this layer.

```
class MathServiceWSRF(MathService):

    def GetResourceContext(ps, address):
        """get a resource context"""
        return ManagerHome.getInstance().getResourceContext(ps,
 address)
    GetResourceContext = staticmethod(GetResourceContext)

    ...
    ...

    def wsa_Destroy(self, ps, address, **kw): ❶

 #http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-
1.2-draft-01.wsdl DestroyRequest
        request,response = MathService.wsa_Destroy(self, ps, address)
 ❷

        ctx = MathServiceWSRF.GetResourceContext(ps, address) ❸
        ImmediateResourceTermination.Destroy(ctx) ❹
        return request,response ❺
```

❶ Fully functional WSRF Destroy operation, immediately destroys the instance referenced by the EPR.

❷ Call lower-level service binding stub to set up the `response` and `request` python objects.

❸ Retrieve the `ResourceContext` of the service instance.

❹ Request immediate destruction of the service instance.

❺ Return python object representing the WS-ResourceLifetime [http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-draft-01.wsdl] Message `DestroyResponse`

Sample Client

Note

Download: client.py [http://dsd.lbl.gov/gtg/projects/pyGridWare/doc/downloads/lifetime/client.py]

```
#!/usr/bin/env python
############################################################################
# Automatically generated by wsdl2web.py
# See LBNLCopyright for copyright notice!
############################################################################
from twisted.python import log
from twisted.internet import reactor

import ZSI
from pyGridWare.utility.scripts.client import GetBasicOptParser,
 GetPortKWArgs, SetUp
from generated.MathLTService.stubs import MathLTService as CLIENT


def main(**kw):
    locator = CLIENT.MathServiceLocator() ❶
    port = locator.getMathPort(**kw) ❷

    msg = port.createResource(CLIENT.CreateInputMessage()) ❸
    iport =
 locator.getMathPort(endPointReference=msg.EndpointReference, **kw) ❹
    iport.add(CLIENT.AddInputMessage(100)) ❺
    iport.Destroy(CLIENT.DestroyRequest()) ❻
    try:
        iport.subtract(CLIENT.SubtractInputMessage(99)) ❼
    except ZSI.fault.Fault, ex:
        pass

    reactor.stop()
    print "EXCEPTION: ", ex.__class__ ❽
    print '\tcode   -- ', ex.code
    print '\tstring -- ', ex.string
    print '\targs   -- ', ex.args
    print '\tdetail -- ', ex.detail
    print '\tactor  -- ', ex.actor
    print '\theaderdetail -- ', ex.headerdetail
    print
    print "ZSI Fault:\n", ex.detail[0]
    print

if __name__ == '__main__':
    op = GetBasicOptParser()
    (options, args) = op.parse_args()
    SetUp(options)
    kw = GetPortKWArgs(options)
    reactor.callWhenRunning(main, **kw)
    reactor.run()
```

1. Here we obtain a reference to a `MathPortType` instance. Notice the factory's URI, where the service is located, and the WS-Addressing URI, which specifies the WS-Addressing version the client should use.
2. Once we have the MathPortType, we use it to invoke the `createResource` operation. This operation returns an endpoint reference, inside a `CreateResourceResponse` object. This endpoint reference includes both the instance service's URI *and* the new resource's identifier. In the next client we will take a peek inside the endpoint reference.
3. Call the `createResource` factory method, this returns a CreateOutputMessage instance.
4. The CreateOutputMessage contains an EPR, use it to obtain a new `MathPortType` which will now refer to the instance service.
5. We now use the `iport` to invoke `add`, this returns successfully.
6. We now use the `iport` to invoke `destroy`, which immediately destroys the resource.
7. Any calls made by the `iport` will fail now since the resource has been destroyed. Invoking `subtract` fails, the server returns a SOAP Fault containing a WSRP ResourceUnknownFault [].
8. `ZSI.fault.Fault`, SOAP Fault Exception.

Run the client:

Note

Make sure to start the server first...

```
% ./client.py -u http://127.0.0.1:9080/wsrf/services/MathService -d 0

EXCEPTION:  ZSI.fault.Fault
        code   --  (u'http://schemas.xmlsoap.org/soap/envelope/',
 u'Server')
        string --  Processing Failure
        args   --  ((u'http://schemas.xmlsoap.org/soap/envelope/',
 u'Server'), u'Processing Failure', None, [>ZSI.fault.ZSIFaultDetail
 o140ecb0<], None)
        detail --  [>ZSI.fault.ZSIFaultDetail o140ecb0<]
        actor  --  None
        headerdetail --  None

ZSI Fault:
pyGridWare.wsrf.faults.PropertiesFaults:ResourceUnknownFault
<pyGridWare.wsrf.faults.PropertiesFaults.pyGridWare.wsrf.faults.PropertiesFaults.R
 instance 16388a0
<ns2:ResourceUnknownFault
 xmlns:ns1="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
BaseFaults-1.2-draft-01.xsd"
 xmlns:ns2="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties-1.2-draft-01.xsd"><ns1:Timestamp
 xmlns:ns3="http://www.w3.org/2001/XMLSchema"
 xmlns:ns4="http://www.w3.org/2001/XMLSchema-instance"
 ns4:type="ns3:dateTime">2006-04-14T21:27:52Z</ns1:Timestamp></
ns2:ResourceUnknownFault>>
[trace:
 /Users/boverhof/Desktop/Workspace/Python/zsi/ZSI/twisted/
WSresource.py:341:render_POST
```

```
/Users/boverhof/Desktop/Workspace/Python/zsi/ZSI/twisted/
WSresource.py:254:processRequest
/Library/Frameworks/Python.framework/Versions/2.4/lib/python2.4/site-
packages/pyGridWare-1.2.0rc3-py2.4.egg/pyGridWare/utility/web/
resource.py:77:processRequest
/private/tmp/MathLTSite/services/MathService.rpy:19:wsa_subtract
/private/tmp/MathLTSite/generated/MathLTService/services/
MathLTService/MathService.py:19:GetResourceContext
/private/tmp/MathLTSite/generated/MathLTService/resource/
MathLTService/MathService.py:24:getResourceContext
/Library/Frameworks/Python.framework/Versions/2.4/lib/python2.4/site-
packages/pyGridWare-1.2.0rc3-py2.4.egg/pyGridWare/resource/
ResourceHome.py:52:get]
```

When `subtract` is invoked, the endpoint reference that is in the call to `GetResourceContext` refers to a resource that no longer exists, thus a `ResourceUnknownFault` is thrown and returned in a SOAP fault exception (`ZSI.fault.Fault`) to the client.

Scheduled destruction

Scheduled destruction is a more elaborate form of resource lifecycle management, as it allows us to specify exactly when we want the resource to be destroyed. The main application of scheduled destruction is to perform *lease-based lifecycle management*, where we initially set the destruction time of a resource some time in the future (for example, 5 minutes). This is called the *lease*. Our application must periodically *renew the lease* (setting the destruction time another 5 minutes in the future), or the resource will eventually be destroyed. This will allow our application to purge resources that for some reason (network failure, programmer errors, etc.) have become unavailable (and therefore can't receive the lease renewal).

Using scheduled destruction requires adding more code that immediate destruction because the standard WSRF portType that provides scheduled destruction not only adds a new operation (`SetTerminationTime`) but also two new resource properties: `TerminationTime` and `CurrentTime`. `TerminationTime` specifies when the resource is set to be destroyed, and the value of `CurrentTime` must always be the time in the machine that hosts the resource. This means that, not only will we have to modify the WSDL file, we will also have to make sure those two new resource properties are properly implemented in our resource class.

Overview: Directions to add scheduled destruction to our service

Note

The WSDL presented in the last section of this chapter describes a service with WS-ResourceLifetime properties. You can skip the next section, in which I present a high level explanation of how to add scheduled destruction to the math service.

We need to add the standard WSRF `ScheduledResourceTermination` portType to our service through compositional inheritance. This portType adds a `SetTerminationTime` operation to our portType that will instruct the current resource to terminate itself at the specified time. In sum, we need to add five things to our WSDL. First, copy the `ScheduledResourceTermination` portType

`SetTerminationTime` operation to the MathPortType. Second, add a `SetTerminationTime` binding operation to MathBinding. Third, add a WSDL import for the WS-ResourceLifetime [] WSDL. Forth, add a XSD import for the WS-ResourceLifetime [] schema. Lastly, add the two global element declarations compromising the `ScheduledResourceTerminationRP` Resource Property to the MathResourceProperties element.

Note

Download: MathLTService.wsdl [http://dsd.lbl.gov/gtg/projects/pyGridWare/doc/downloads/lifetime/scheduled/MathLTService.wsdl]

```
% curl -O
 http://dsd.lbl.gov/gtg/projects/pyGridWare/doc/downloads/lifetime/
MathLTService.wsdl
% wsdl2web --force MathLTService.wsdl
```

Generated Stub Service: pyGridWare.generated.services.math.MathService

Note

This class has been generated for you.

```
class MathServiceWSRF(MathService):

    def GetResourceContext(ps, address):
        """get a resource context"""
        return ManagerHome.getInstance().getResourceContext(ps,
 address)
    GetResourceContext = staticmethod(GetResourceContext)

    ...
    ...

    def wsa_SetTerminationTime(self, ps, address, **kw): ❶

 #http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-
1.2-draft-01.wsdl SetTerminationTimeRequest
        request,response = MathService.wsa_SetTerminationTime(self,
 ps, address) ❷
        ctx = MathServiceWSRF.GetResourceContext(ps, address) ❸
        ScheduledResourceTermination.SetTerminationTime(ctx,
 request._RequestedTerminationTime) ❹
        response._NewTerminationTime =
 time.gmtime(ctx.getTerminationTime()) ❺
        response._CurrentTime = time.gmtime(time.time())
        return request,response ❻
```

1. Fully functional WSRF SetTerminationTime operation. Schedules a service instance, referenced by the EPR, for destruction.
2. Call lower-level service binding stub to set up the `response` and `request` python objects.
3. Retrieve the `ResourceContext` of the service instance.
4. Schedule the destruction of the service instance represented by the `ResourceContext`.
5. The `response` variable is an instance of the generated class `SetTerminationTimeResponse` representing the WSDL SetTerminationTimeResponse Message. The attributes representing the `NewTerminationTime` and `CurrentTime` need to be set before returning this "message" to the client.
6. Return python object representing the WS-ResourceLifetime [http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-draft-01.wsdl] Message `SetTerminationTimeResponse`

client_set_termination_time.py

We will test our service by creating a new resource, setting its termination 10 seconds in the future, and then checking every second to see if the resource is still 'alive'. When the resource is terminated, any call to the resource will produce an exception. Like the immediate destruction client, this client is similar to the simple client seen in ????. The following is the code that we will run after the resource has been created:

Note

Download: client_set_termination_time.py [http://dsd.lbl.gov/gtg/projects/pyGridWare/doc/downloads/lifetime/scheduled/client_set_termination_time.py]

```
#!/usr/bin/env python
############################################################################
# Automatically generated by wsdl2web.py
# See LBNLCopyright for copyright notice!
############################################################################
from twisted.python import log
from twisted.internet import reactor

import ZSI
from pyGridWare.utility.scripts.client import GetBasicOptParser,
 GetPortKWArgs, SetUp
from generated.MathLTService.stubs import MathLTService as CLIENT


def main(**kw):
    import time
    locator = CLIENT.MathServiceLocator() ❶
    port = locator.getMathPort(**kw) ❷

    msg = port.createResource(CLIENT.CreateInputMessage()) ❸
    print 'Created instance.'

    iport =
 locator.getMathPort(endPointReference=msg.EndpointReference, **kw) ❹

    request = CLIENT.SetTerminationTimeRequest()
```

```
    startTime = time.time()
    termTime = startTime + 10

    request.RequestedTerminationTime = termTime ❺

    msg = iport.SetTerminationTime(request) ❻

    print 'Local Time'
    print 'Start Time                 ', time.strftime('%c',
 time.localtime(startTime))
    print 'Requested termination time ', time.strftime('%c',
 time.localtime(termTime))
    print
    print 'UTC Tuples'
    print 'RequestedTerminationTime ',
 time.gmtime(request.RequestedTerminationTime)
    print 'NewTerminationTime       ', msg.NewTerminationTime
    print 'CurrentTime              ', msg.CurrentTime
    print

    try:
        for i in range(1,20):

            iport.add(CLIENT.AddInputMessage(1)) ❼
            time.sleep(startTime + i - time.time())
            print 'Second %d' %i

    except ZSI.fault.Fault:❽
        print 'resource has been destroyed: ', time.strftime('%c',
 time.localtime())
    else:
        print 'ERROR'

    reactor.stop()


if __name__ == '__main__':
    op = GetBasicOptParser()
    (options, args) = op.parse_args()
    SetUp(options)
    kw = GetPortKWArgs(options)
    reactor.callWhenRunning(main, **kw)
    reactor.run()
```

❶ Here we obtain a reference to a `MathPortType` instance. Notice the factory's URI, where the service is located, and the WS-Addressing URI, which specifies the WS-Addressing version the client should use.

❷ Once we have the MathPortType, we use it to invoke the `createResource` operation. This operation returns an endpoint reference, inside a `CreateResourceResponse` object. This endpoint reference includes both the instance service's URI *and* the new resource's identifier. In the next client we will take a peek inside the endpoint reference.

❸ Call the `createResource` factory method, this returns a CreateOutputMessage instance.

❹ The CreateOutputMessage contains an EPR, use it to obtain a new `MathPortType` which will now refer to the instance service.

6

6 We now use the `iport` to invoke `SetTerminationTime`, which schedules the destruction of the resource.

7 Invoke the add operation with `iport` until the the server returns a SOAP Fault containing a WSRP ResourceUnknownFault [].

8 `ZSI.fault.Fault`, SOAP Fault Exception contains a ResourceUnknownFault, it has been destroyed.

Run the client:

Note

Make sure to start the server first...

```
% ./client_set_termination_time.py -u
 http://127.0.0.1:9080/wsrf/services/MathService -d 0
Created instance.
Local Time
Start Time                   Fri Apr 14 14:49:44 2006
Requested termination time  Fri Apr 14 14:49:54 2006

UTC Tuples
RequestedTerminationTime  (2006, 4, 14, 21, 49, 54, 4, 104, 0)
NewTerminationTime        (2006, 4, 14, 21, 49, 54, 0, 0, 0)
CurrentTime               (2006, 4, 14, 21, 49, 43, 0, 0, 0)

Second 1
Second 2
Second 3
Second 4
Second 5
Second 6
Second 7
Second 8
Second 9
Second 10
Second 11
Second 12
'HTTP Error 500'
resource has been destroyed:  Fri Apr 14 14:49:56 2006
```

Immediate and Scheduled Termination MathService WSDL

Note

Download: MathLTService.wsdl [http://dsd.lbl.gov/gtg/projects/pyGridWare/doc/downloads/lifetime/MathLTService.wsdl]

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<definitions name="MathServiceRP"

 targetNamespace="http://www.globus.org/namespaces/examples/core/
MathService_instance_rp"
    xmlns="http://schemas.xmlsoap.org/wsdl/"

 xmlns:tns="http://www.globus.org/namespaces/examples/core/
MathService_instance_rp"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"

 xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties-1.2-draft-01.xsd"

 xmlns:wsrpw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties-1.2-draft-01.wsdl"

 xmlns:wsrlw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceLifetime-1.2-draft-01.wsdl"❶
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<wsdl:import
    namespace=

 "http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-
1.2-draft-01.wsdl"

 location="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties-1.2-draft-01.wsdl" />


<wsdl:import ❷

    namespace=
    "http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-
1.2-draft-01.wsdl"

 location="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceLifetime-1.2-draft-01.wsdl" />



<!--============================================================

                    T Y P E S

  ============================================================-->
<types>
<xsd:schema
 targetNamespace="http://www.globus.org/namespaces/examples/core/
MathService_instance_rp"
```

```
 xmlns:tns="http://www.globus.org/namespaces/examples/core/
MathService_instance_rp"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"

 xmlns:wsrl="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceLifetime-1.2-draft-01.xsd"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

        <!-- IMPORT WS-ADDRESSING -->
        <xsd:import
 namespace="http://schemas.xmlsoap.org/ws/2004/03/addressing"

 schemaLocation="http://schemas.xmlsoap.org/ws/2004/03/addressing"/>

        <!-- IMPORT WS-ResourceLifetime schemas -->

        <xsd:import
 namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceLifetime-1.2-draft-01.xsd" (3)


 schemaLocation="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceLifetime-1.2-draft-01.xsd"/>

        <!-- REQUESTS AND RESPONSES -->

        <xsd:element name="add" type="xsd:int"/>
        <xsd:element name="addResponse">
                <xsd:complexType/>
        </xsd:element>
        <xsd:element name="subtract" type="xsd:int"/>
        <xsd:element name="subtractResponse">
                <xsd:complexType/>
        </xsd:element>

        <xsd:element name="create">
                <xsd:complexType/>
        </xsd:element>
        <xsd:element name="createResponse">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element ref="wsa:EndpointReference"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>

        <!-- MathService Resource Properties -->

        <xsd:element name="Value" type="xsd:int"/>
        <xsd:element name="LastOp" type="xsd:string"/>
        <xsd:element name="MathResourceProperties">
            <xsd:complexType>
```

```
                <xsd:sequence>
                    <xsd:element ref="tns:Value"/>
                    <xsd:element ref="tns:LastOp"/>

                    <xsd:element maxOccurs="1" minOccurs="1"
 ref="wsrl:CurrentTime"/> ❹

                    <xsd:element maxOccurs="1" minOccurs="1"
 ref="wsrl:TerminationTime"/> ❻
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>

</xsd:schema>
</types>

<!--================================================================

                        M E S S A G E S

  ================================================================-->
<message name="AddInputMessage">
        <part name="parameters" element="tns:add"/>
</message>
<message name="AddOutputMessage">
        <part name="parameters" element="tns:addResponse"/>
</message>

<message name="SubtractInputMessage">
        <part name="parameters" element="tns:subtract"/>
</message>
<message name="SubtractOutputMessage">
        <part name="parameters" element="tns:subtractResponse"/>
</message>

<message name="CreateInputMessage">
        <part name="parameters" element="tns:create"/>
</message>
<message name="CreateOutputMessage">
        <part name="parameters" element="tns:createResponse"/>
</message>


<!--================================================================

                        P O R T T Y P E

  ================================================================-->

<portType name="MathPortType"
    wsrp:ResourceProperties="tns:MathResourceProperties">

 <operation name="add">
```

```
  <input message="tns:AddInputMessage"/>
  <output message="tns:AddOutputMessage"/>
 </operation>

 <operation name="subtract">
  <input message="tns:SubtractInputMessage"/>
  <output message="tns:SubtractOutputMessage"/>
 </operation>

 <operation name="createResource">
  <input message="tns:CreateInputMessage"/>
  <output message="tns:CreateOutputMessage"/>
 </operation>

        <!-- WS-ResourceProperties operations -->
 <operation name="GetResourceProperty">
  <input name="GetResourcePropertyRequest"
                    message="wsrpw:GetResourcePropertyRequest"

 wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties/GetResourceProperty"/>
                <output name="GetResourcePropertyResponse"
                    message="wsrpw:GetResourcePropertyResponse"

 wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties/GetResourcePropertyResponse"/>
                <fault name="InvalidResourcePropertyQNameFault"
                    message="wsrpw:InvalidResourcePropertyQNameFault"/
>
                <fault name="ResourceUnknownFault"
                    message="wsrpw:ResourceUnknownFault"/>
 </operation>

    <operation name="GetMultipleResourceProperties">
      <input name="GetMultipleResourcePropertiesRequest"
 message="wsrpw:GetMultipleResourcePropertiesRequest"
 wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties/GetMultipleResourceProperties"/>
      <output name="GetMultipleResourcePropertiesResponse"
 message="wsrpw:GetMultipleResourcePropertiesResponse"
 wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties/GetMultipleResourcePropertiesResponse"/>
      <fault name="InvalidResourcePropertyQNameFault"
 message="wsrpw:InvalidResourcePropertyQNameFault"/>
      <fault name="ResourceUnknownFault"
 message="wsrpw:ResourceUnknownFault"/>
    </operation>

    <operation name="SetResourceProperties">
      <input name="SetResourcePropertiesRequest"
 message="wsrpw:SetResourcePropertiesRequest" wsa:Action="http://
docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties/
SetResourceProperties"/>
```

```
      <output name="SetResourcePropertiesResponse"
 message="wsrpw:SetResourcePropertiesResponse" wsa:Action="http://
docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties/
SetResourcePropertiesResponse"/>
      <fault name="InvalidResourcePropertyQNameFault"
 message="wsrpw:InvalidResourcePropertyQNameFault"/>
      <fault name="UnableToModifyResourcePropertyFault"
 message="wsrpw:UnableToModifyResourcePropertyFault"/>
      <fault name="SetResourcePropertyRequestFailedFault"
 message="wsrpw:SetResourcePropertyRequestFailedFault"/>
      <fault name="ResourceUnknownFault"
 message="wsrpw:ResourceUnknownFault"/>
      <fault name="InvalidSetResourcePropertiesRequestContentFault"
 message="wsrpw:InvalidSetResourcePropertiesRequestContentFault"/>
    </operation>

    <operation name="QueryResourceProperties">
      <input name="QueryResourcePropertiesRequest"
 message="wsrpw:QueryResourcePropertiesRequest" wsa:Action="http://
docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties/
QueryResourceProperties"/>
      <output name="QueryResourcePropertiesResponse"
 message="wsrpw:QueryResourcePropertiesResponse" wsa:Action="http://
docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties/
QueryResourcePropertiesResponse"/>
      <fault name="UnknownQueryExpressionDialectFault"
 message="wsrpw:UnknownQueryExpressionDialectFault"/>
      <fault name="InvalidResourcePropertyQNameFault"
 message="wsrpw:InvalidResourcePropertyQNameFault"/>
      <fault name="QueryEvaluationErrorFault"
 message="wsrpw:QueryEvaluationErrorFault"/>
      <fault name="InvalidQueryExpressionFault"
 message="wsrpw:InvalidQueryExpressionFault"/>
      <fault name="ResourceUnknownFault"
 message="wsrpw:ResourceUnknownFault"/>
    </operation>

    <!-- WS-ResourceLifetime operations -->
    <!--     ScheduledResourceTermination -->
        <wsdl:operation name="SetTerminationTime">

            <wsdl:input message="wsrlw:SetTerminationTimeRequest"

 wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceLifetime/SetTerminationTime"/>
            <wsdl:output message="wsrlw:SetTerminationTimeResponse"

 wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceLifetime/SetTerminationTimeResponse"/>
            <wsdl:fault
                message="wsrlw:UnableToSetTerminationTimeFault"
                name="UnableToSetTerminationTimeFault"/>
```

```
            <wsdl:fault message="wsrlw:ResourceUnknownFault"
 name="ResourceUnknownFault"/>
            <wsdl:fault
                message="wsrlw:TerminationTimeChangeRejectedFault"
                name="TerminationTimeChangeRejectedFault"/>
        </wsdl:operation>

    <!--   ImmediateResourceTermination -->
        <wsdl:operation name="Destroy">

            <wsdl:input message="wsrlw:DestroyRequest"

 wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceLifetime/Destroy"/>
            <wsdl:output message="wsrlw:DestroyResponse"

 wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceLifetime/DestroyResponse"/>
            <wsdl:fault
                message="wsrlw:ResourceNotDestroyedFault"
                name="ResourceNotDestroyedFault"/>
            <wsdl:fault message="wsrlw:ResourceUnknownFault"
 name="ResourceUnknownFault"/>
        </wsdl:operation>

</portType>

<!--================================================================

                        B I N D I N G

  ================================================================-->
<binding name="MathBinding" type="MathPortType">
    <soap:binding style="document"
 transport="http://schemas.xmlsoap.org/soap/http"/>

    <wsdl:operation name="add">
      <soap:operation soapAction=""/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>

    <wsdl:operation name="subtract">
      <soap:operation soapAction=""/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
```

```
  </wsdl:output>
</wsdl:operation>

<wsdl:operation name="createResource">
  <soap:operation soapAction=""/>
  <wsdl:input>
    <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal"/>
  </wsdl:output>
</wsdl:operation>

<wsdl:operation name="GetResourceProperty">
  <soap:operation soapAction=""/>
  <wsdl:input>
    <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal"/>
  </wsdl:output>
</wsdl:operation>

<wsdl:operation name="GetMultipleResourceProperties">
  <soap:operation soapAction=""/>
  <wsdl:input>
    <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal"/>
  </wsdl:output>
</wsdl:operation>

<wsdl:operation name="SetResourceProperties">
  <soap:operation soapAction=""/>
  <wsdl:input>
    <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal"/>
  </wsdl:output>
</wsdl:operation>

<wsdl:operation name="QueryResourceProperties">
  <soap:operation soapAction=""/>
  <wsdl:input>
    <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal"/>
  </wsdl:output>
</wsdl:operation>
```

```
    <wsdl:operation name="SetTerminationTime">

      <soap:operation soapAction=""/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>

    <wsdl:operation name="Destroy">
      <soap:operation soapAction=""/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>


</binding>

<!--============================================================

                      S E R V I C E

  ============================================================-->
<service name="MathService">
    <port name="MathPort" binding="MathBinding">
       <soap:address location="http://localhost:8080/wsrf/services/"/>
    </port>
</service>

</definitions>
```

Chapter 7. Notifications

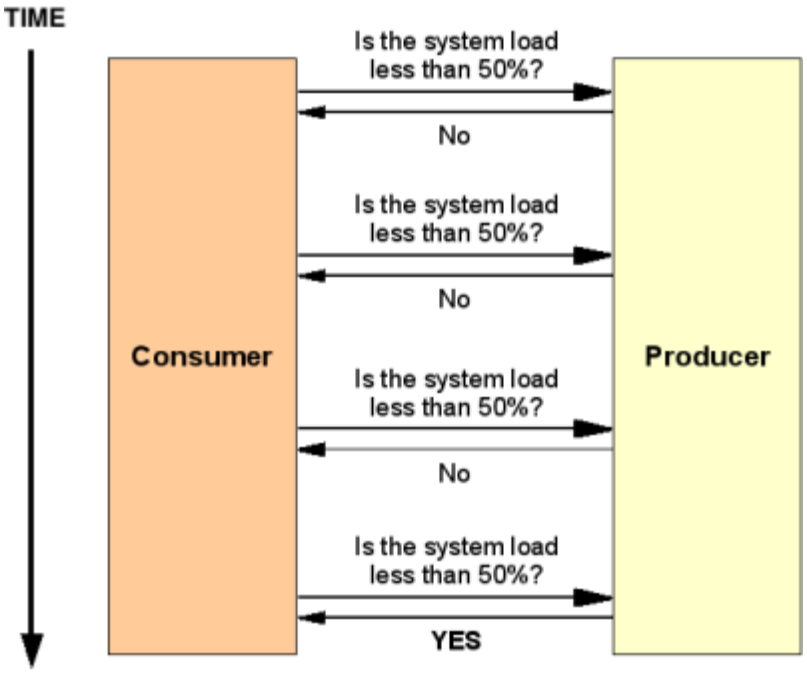
In this chapter we will introduce the concepts of `notification`, a common design pattern that allows clients to be notified when interesting events happen in a server. In particular, we will focus on WS-Notifications, a family of specification that allow us to use this design pattern with Web Services. Then, we will see two examples of how we can use notifications in our services.

What are notifications?

Notifications are nothing new. It's a very popular software design pattern, although you might know it with a different name such as Observer/Observable. Let's suppose that our software had several distinct parts (e.g. a GUI and the application logic, a client and a server, etc.) and that one of the parts of the software needs to be aware of the changes that happen in one of the other parts. For example, the GUI might need to know when a value is changed in a database, so that the new value is immediately displayed to the user. Taking this to the client/server world is easy: suppose a client needs to know when the server reaches a certain state, so the client can perform a certain action.

The most crude approach to keep the client informed is a *polling* approach. The client periodically *polls* the server (asks if there are any changes). For example, let's suppose a client applications wants to know when the load of a server drops below 50%. The server is called the *producer* of events (in this case, the event is a drop in the server load). The client, on the other hand, is called the *consumer* of events. The polling approach would go like this:

Figure 7.1. Keeping track of changes using polling



1. The consumer asks the producer if there are any changes. The producer replies "No", so the consumer waits a while before making another call.

2. Once again, the consumer asks the producer if there are any changes. The producer replies "No", so the consumer waits a while before making another call.

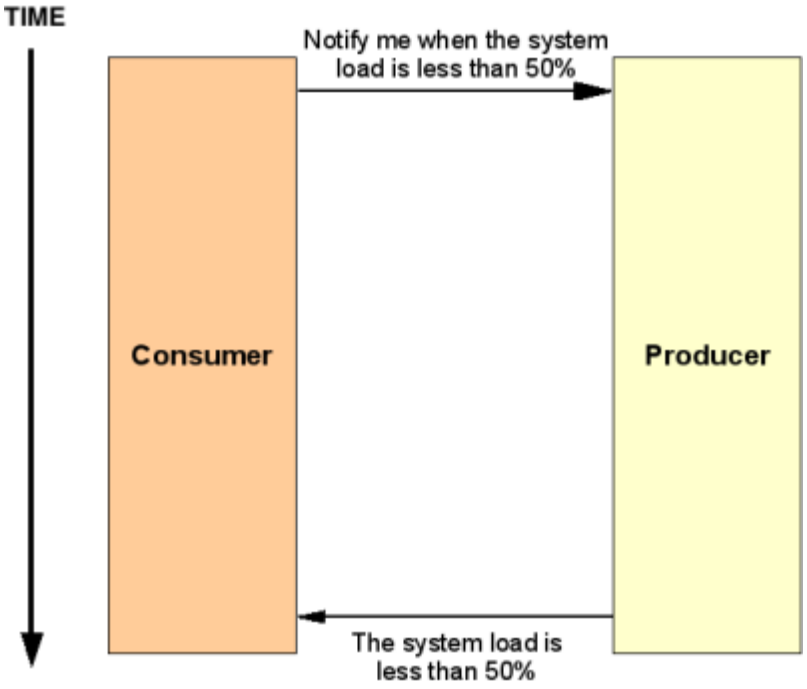
3. As you can see, this step can be repeated *ad nauseam* until the server finally replies that there has been a change.

This approach isn't very efficient, specially if you consider the following:

- If the time between calls is very small, the amount of network traffic and CPU use increases.
- There can be more than one consumer. If we have dozens of consumers, waiting for an event to happen, then the producer could get saturated with calls asking it if there are any changes.

The answer to this problem is actually terribly simple (and common sense). Instead of periodically asking the producer if there are any changes, we make an initial call asking the producer to *notify* the consumer whenever a certain event occurs. This is the *notification* approach.

Figure 7.2. Keeping track of changes using notifications



1. The consumer asks the producer to notify him as soon as the server load drops below 50%. The producer keeps a list of all its registered consumers. This step is normally called the *subscription* or *registration* step.
2. The consumer and the producer go about their business until the server load drops below 50%.
3. Once the server load drops below 50%, the producer *notifies* all its consumers (remember, there can be more than one) of that event.

As you can see, this approach is much more efficient (in this simple example, network traffic has been sliced in half with respect to the polling approach).

WS-Notifications

The WS-Notifications family of specifications, although not a part of WSRF, has strong ties to it. It provides a set of standard interfaces to use the notification design pattern with Web Services.

WS-Notifications is divided into three specifications: WS-Topics, WS-BaseNotification, and WS-BrokeredNotification.

WS-Topics

First of all, we have *topics*, which are used by the other two specifications in WS-Notifications to present a set of "items of interest for subscription". As we will see next, a service can publish a set of topics that clients can subscribe to, and receive a notification whenever the topic changes. Topics are very versatile, as they even allow us to create *topic trees*, where a topic can have a set of *child topics*. By subscribing to a topic, a client automatically receives notifications from all the descendant topics (without having to manually subscribe to each of them).

WS-BaseNotification

This specification defines the standard interfaces of notification consumers and producers. In a nutshell, notification producers have to expose a *subscribe* operation that notification consumers can use to request a subscription. Consumers, in turn, have to expose a *notify* operation that producers can use to deliver the notification. Furthermore, the client actually requesting the subscription need not necessarily be the consumer of those notifications. In other words, clients can perform subscriptions "on behalf of other notification consumers".

Figure 7.3. A WS-Notification interaction where the subscriber and the consumer are different entities

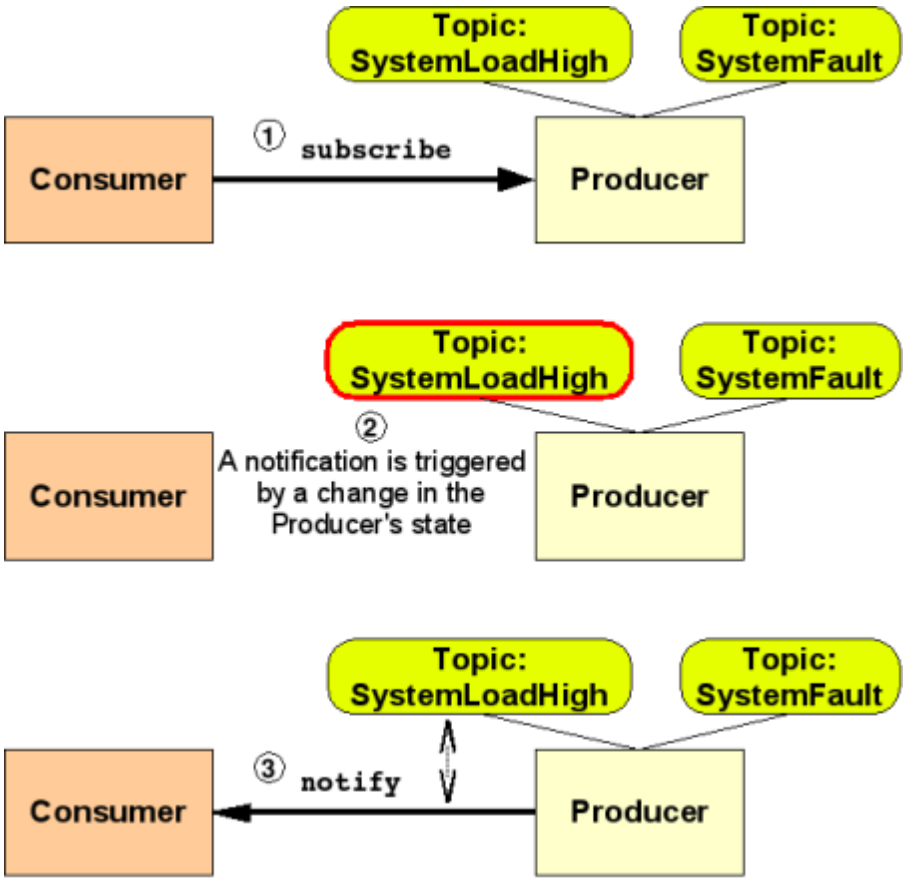


Figure 7.3, “Another typical WS-Notification interaction” shows an example interaction between a notification consumer and producer, in the simple case when the subscriber and consumer are the same entity. In this example we have a single notification consumer, and a single notification producer that publishes two topics: `SystemLoadHigh` and `SystemFault`.

1. First of all, the notification consumer subscribes himself to the `SystemLoadHigh` topic. It is interesting to note that, internally, a `Subscription` resource is created with information regarding the subscription (not shown in the figure).

2. Next, at some point in time, something happens in the notification producer that must trigger a notification from the `SystemLoadHigh` topic. For example, we might have implemented our service to send out a notification every time the system load passes from "more than 50%" to "less than 50%".

3. The notification producer delivers the notification to the consumer by invoking the `notify` operation in the consumer. As shown in the figure, this notification delivery is tied to the topic that triggered the notification.

Figure 7.4. A WS-Notification interaction where the subscriber and the consumer are different entities

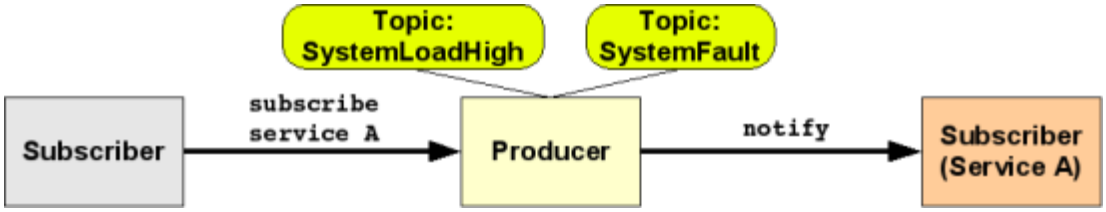


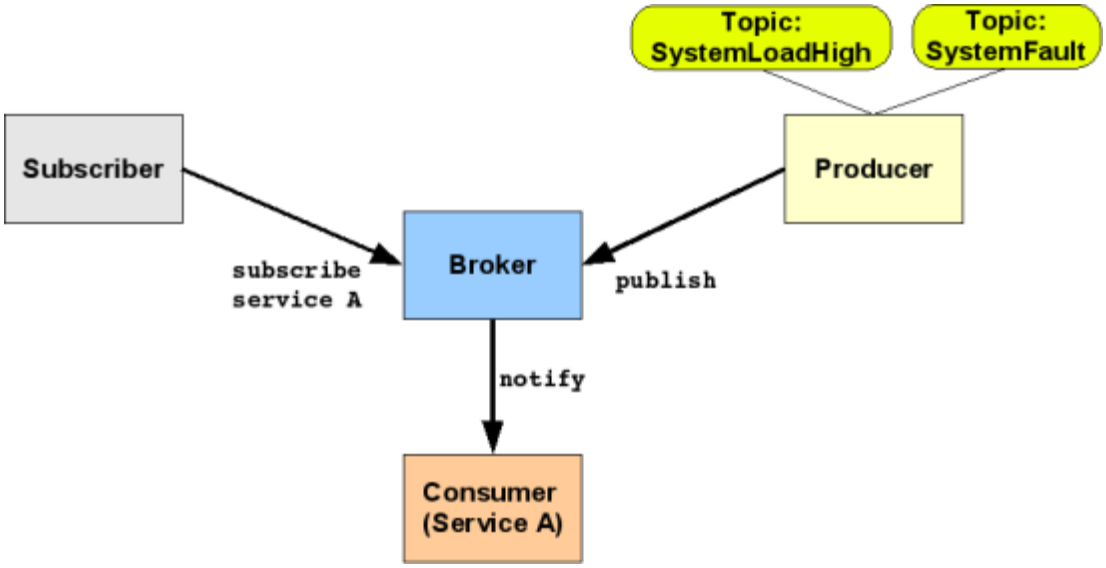
Figure 7.4, “Another typical WS-Notification interaction”, on the other hand, shows how the subscriber and the consumer need not be the same entity. In the figure, the subscriber requests the producer that Service A be subscribed to the `SystemLoadHigh` topic. When a notification is triggered, the notification is sent to Service A (the consumer) not to the subscriber.

WS-BrokeredNotification

In brokered notifications we consider the case when notifications are delivered from the producer to the consumer through an intermediate entity called the *broker*. The WS-BrokeredNotification defines the standard interfaces for the notification broker.

As shown in Figure 7.5, “A typical brokered WS-Notification interaction”, in the presence of a notification broker, the producer must register with the broker and publish its topics there. The subscriber (separate from the consumer in this case), must also subscribe through the broker, not directly with the producer. Finally, when a notification is produced, it is delivered to the consumer through the broker.

Figure 7.5. A typical brokered WS-Notification interaction



Notifications in GT4

GT4 currently doesn't implement the WS-Notifications family of specifications completely. For example, no support for brokered notification is included. However, GT4 does allow us to perform effective topic-based notification. One of the more interesting parts of the GT4 implementation of WS-Notifications is that it will allow us to effortlessly expose a resource property as a topic, triggering a notification each time the value of the RP changes. We will also be able to define our own topics, which need not trigger a notification every single time the value of an RP changes. In the remainder of the chapter, we will see how we can add both types of topics to our service.

Notifying changes in a resource property

We will see how we can add notifications to a service so clients can be notified each time a certain RP is modified. As we did in ???? and ????, our example will be based, for simplicity, on the `SimpleResourceHome` resource home.

Create a New Site with a NotificationProducer Service

Our portType will need to extend from a standard WS-Notifications portType called `NotificationProducer`, which exposes a `Subscribe` operation that consumers can use to subscribe themselves to a particular topic. Since this examples takes an existing resource property and exposes it as a topic, no additional WSDL code is required beyond extending the `NotificationProducer` portType.

Note

Download the MathNPService.wsdl [http://dsd.lbl.gov/gtg/projects/pyGridWare/doc/downloads/notification/MathNPService.wsdl], it is also at the end of this chapter.

```
% mkdir /tmp/MathNPSite
% cd /tmp/MathNPSite
```

```
% curl -O
 http://dsd.lbl.gov/gtg/projects/pyGridWare/doc/downloads/
notification/MathNPService.wsdl
% wsdl2web --script=notification_client.py --rpy=MathService.rpy
 MathNPService.wsdl
% cat /tmp/MathLTSite/services/MathService.rpy | sed
 s/MathLTService/MathNPService/g > services/MathService.rpy
```

Note

Site is finished!

Generated WSRF Service: class MathServiceWSRF

Generated WSRF Service: class MathServiceWSRF

class MathServiceWSRF has been generated for you, it implements the WS-BaseNotification NotificationProducer operations. Module path is `generated.MathNPService.services.MathNPService.MathService.MathServiceWSRF`.

```
class MathServiceWSRF(MathService):

    def GetResourceContext(ps, address):
        """get a resource context"""
        return ManagerHome.getInstance().getResourceContext(ps,
 address)
    GetResourceContext = staticmethod(GetResourceContext)

    ...
    ...

    def wsa_Subscribe(self, ps, address, **kw): ❶

 #http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-
draft-01.wsdl SubscribeRequest
        request,response = MathService.wsa_Subscribe(self, ps,
 address) ❷
        ctx = self.GetResourceContext(ps, address) ❸
        assert
 SubscriptionManager._isInstantiated(),"SubscriptionManager is not
 instantiated"
        manager = SubscriptionManager.getInstance() ❹
        response._SubscriptionReference = manager.subscribe(request,
 ctx) ❺
        return request,response ❻
```

❶ Subscribe to Notifications via the SubscriptionManager/NotificationProducer

❷ Call lower-level service binding stub to return `SubscribeRequest` and `SubscribeResponse` instances.

❸ Retrieve the `ResourceContext` of the service instance.

4. Retrieve the `SubscriptionManager` Singleton.
5. Use the `SubscriptionManager` to subscribe, returning an endpoint reference which is used as the `SubscriptionReference`.
6. Return python object representing the `SubscribeResponse`

Using the NotificationConsumer Client

Note

Download: notification_client.py [http://dsd.lbl.gov/gtg/projects/pyGridWare/doc/downloads/notification/notification_client.py]

```
#!/usr/bin/env python
############################################################################
# Automatically generated by wsdl2web.py
# See LBNLCopyright for copyright notice!
############################################################################
from twisted.python import log
from twisted.internet import reactor

import ZSI
from ZSI.fault import Fault
from pyGridWare.utility.scripts.client import GetBasicOptParser,
 GetPortKWArgs, SetUp

from pyGridWare.wsrf.notification import SubscribeUtility
from pyGridWare.wsrf.notification.NotificationConsumer import \
    NotificationConsumer, NotificationConsumerService

from pyGridWare.wsrf.faults.PropertiesFaults import
 ResourceUnknownFault

from generated.MathNPService.stubs import MathNPService as CLIENT


def failure(iport):
    print "Attempt to access Destroyed Resource"
    try:
        msg = iport.add(CLIENT.AddInputMessage(1))
    except Fault, ex:
        print "RESOURCE DESTROYED..", ex (1)
        print '\tcode   -- ', ex.code
        print '\tstring -- ', ex.string
        print '\tdetail'
        for d in ex.detail:
            print '\t\tstring -- %s'  %d.string
            print '\t\ttrace  -- %s'  %d.trace

    except Fault, ex:
        print "Unexpected SOAP Fault...", ex.__class__
```

```
    reactor.stop()


def main(**kw):
    target = 10
    locator = CLIENT.MathServiceLocator()
    port = locator.getMathPort(**kw)
    msg = port.createResource(CLIENT.CreateInputMessage()) ❷
    print 'Created instance.'

    iport =
 locator.getMathPort(endPointReference=msg.EndpointReference, **kw) ❸

    class _NCService(NotificationConsumerService):❹

        def wsa_Notify(self, ps, address, **kw):❺
            request,response =
 NotificationConsumerService.wsa_Notify(self, ps, address)
            pyobj = request.NotificationMessage[0].Message
            value =
 pyobj.ResourcePropertyValueChangeNotification.NewValue.Any

            print "NewValue", value
            if
 pyobj.ResourcePropertyValueChangeNotification.NewValue.Any !=
 target:❻
                msg = iport.add(CLIENT.AddInputMessage(1))
                return request,response

            print 'Destroy instance.'
            msg = iport.Destroy(CLIENT.DestroyRequest())
            reactor.callLater(0.1, failure, iport)
            return request,response


    consumer = NotificationConsumer(notificationClass=_NCService) ❼
    consumer.start()

    request = CLIENT.SubscribeRequest()
    SubscribeUtility.SimpleTopicExpression(request, consumer, ❽

 "http://www.globus.org/namespaces/examples/core/
MathService_instance_rp", "Value")
    msg = iport.Subscribe(request) ❾
    msg = iport.add(CLIENT.AddInputMessage(1)) ❶❶


if __name__ == '__main__':
    op = GetBasicOptParser()
```

```
    (options, args) = op.parse_args()
    SetUp(options)
    kw = GetPortKWArgs(options)
    reactor.callWhenRunning(main, **kw)
    reactor.run()
```

❷ Create the service instance.

❸ Create a port for communicating with the service instance.

❹ Define a callback class for the notification consumer.

❼ Create a `NotificationConsumer`, pass in the callback class as a parameter.

❽ Use the `SubscribeUtility` to set up a `SimpleTopicExpression`. This subscription specifies that the consumer be notified whenever the `Value` resource property changes.

❾ Use the port to invoke the `Subscribe` operation.

❿ Use the port to invoke the `add` operation. Control is passed to the reactor.

❺ The `Notify` message arrives.

❻ Does the `NewValue` of the `Value` resource property match the target value? If not invoke the `add` operation again, else invoke `Destroy` and queue up the `failure` function.

❶ The service instance is destroyed, so when the `add` operation is invoked a `SOAP:Fault` is caught.

Run the client:

Note

Make sure to start the server first...

```
./notification_client.py -u
 http://127.0.0.1:9080/wsrf/services/MathService -d 0
FtWarning: Creation of InputSource without a URI
/Library/Frameworks/Python.framework/Versions/2.4/lib/python2.4/site-
packages/ZSI-2.0_rc2-py2.4.egg/ZSI/__init__.py:187: SyntaxWarning: The
 null string should be None, not empty.
Created instance.
NewValue 1
NewValue 2
NewValue 3
NewValue 4
NewValue 5
NewValue 6
NewValue 7
NewValue 8
NewValue 9
NewValue 10
Destroy instance.
Attempt to access Destroyed Resource
'HTTP Error 500'
RESOURCE DESTROYED.. Processing Failure
pyGridWare.wsrf.faults.PropertiesFaults:ResourceUnknownFault
<pyGridWare.wsrf.faults.PropertiesFaults.pyGridWare.wsrf.faults.PropertiesFaults.R
 instance 178baf8
```

```
<ns2:ResourceUnknownFault
 xmlns:ns1="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
BaseFaults-1.2-draft-01.xsd"
 xmlns:ns2="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties-1.2-draft-01.xsd"><ns1:Timestamp
 xmlns:ns3="http://www.w3.org/2001/XMLSchema"
 xmlns:ns4="http://www.w3.org/2001/XMLSchema-instance"
 ns4:type="ns3:dateTime">2006-04-15T19:40:37Z</ns1:Timestamp></
ns2:ResourceUnknownFault>>
[trace:
 build/bdist.darwin-8.5.0-Power_Macintosh/egg/ZSI/twisted/
WSresource.py:341:render_POST
build/bdist.darwin-8.5.0-Power_Macintosh/egg/ZSI/twisted/
WSresource.py:254:processRequest
/Library/Frameworks/Python.framework/Versions/2.4/lib/python2.4/site-
packages/pyGridWare-1.2.0rc3-py2.4.egg/pyGridWare/utility/web/
resource.py:77:processRequest
/private/tmp/MathNPSite/services/MathService.rpy:11:wsa_add
/private/tmp/MathNPSite/generated/MathNPService/services/
MathNPService/MathService.py:20:GetResourceContext
/private/tmp/MathNPSite/generated/MathNPService/resource/
MathNPService/MathService.py:24:getResourceContext
/Library/Frameworks/Python.framework/Versions/2.4/lib/python2.4/site-
packages/pyGridWare-1.2.0rc3-py2.4.egg/pyGridWare/resource/
ResourceHome.py:52:get]
        code   --  (u'http://schemas.xmlsoap.org/soap/envelope/',
 u'Server')
        string --  Processing Failure
        detail
                string --
 pyGridWare.wsrf.faults.PropertiesFaults:ResourceUnknownFault
<pyGridWare.wsrf.faults.PropertiesFaults.pyGridWare.wsrf.faults.PropertiesFaults.R
 instance 178baf8
<ns2:ResourceUnknownFault
 xmlns:ns1="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
BaseFaults-1.2-draft-01.xsd"
 xmlns:ns2="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties-1.2-draft-01.xsd"><ns1:Timestamp
 xmlns:ns3="http://www.w3.org/2001/XMLSchema"
 xmlns:ns4="http://www.w3.org/2001/XMLSchema-instance"
 ns4:type="ns3:dateTime">2006-04-15T19:40:37Z</ns1:Timestamp></
ns2:ResourceUnknownFault>>
                trace  --
 build/bdist.darwin-8.5.0-Power_Macintosh/egg/ZSI/twisted/
WSresource.py:341:render_POST
build/bdist.darwin-8.5.0-Power_Macintosh/egg/ZSI/twisted/
WSresource.py:254:processRequest
/Library/Frameworks/Python.framework/Versions/2.4/lib/python2.4/site-
packages/pyGridWare-1.2.0rc3-py2.4.egg/pyGridWare/utility/web/
resource.py:77:processRequest
/private/tmp/MathNPSite/services/MathService.rpy:11:wsa_add
/private/tmp/MathNPSite/generated/MathNPService/services/
MathNPService/MathService.py:20:GetResourceContext
```

```
/private/tmp/MathNPSite/generated/MathNPService/resource/
MathNPService/MathService.py:24:getResourceContext
/Library/Frameworks/Python.framework/Versions/2.4/lib/python2.4/site-
packages/pyGridWare-1.2.0rc3-py2.4.egg/pyGridWare/resource/
ResourceHome.py:52:get
```

What has just happened?

After subscribing the notification consumer is ready to receive notifications about changes to the `Value` resource property. After the initial `add`, control is released to the reactor. Each `add` will cause a `Notify` message to be sent to the notification consumer. In this example I keep adding one to the service instance until I'm notified that the `Value` resource property matches my target value (10). Once this target has been reached, the service instance is destroyed and the function `failure` is queued to test if the service instance is still available.

MathService WSDL NotificationProducer

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="MathServiceRP"

 targetNamespace="http://www.globus.org/namespaces/examples/core/
MathService_instance_rp"
    xmlns="http://schemas.xmlsoap.org/wsdl/"

 xmlns:tns="http://www.globus.org/namespaces/examples/core/
MathService_instance_rp"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"

 xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties-1.2-draft-01.xsd"

 xmlns:wsrpw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties-1.2-draft-01.wsdl"

 xmlns:wsrlw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceLifetime-1.2-draft-01.wsdl"

 xmlns:wsbnw="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-
BaseNotification-1.2-draft-01.wsdl"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<wsdl:import
    namespace=

 "http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-
1.2-draft-01.wsdl"

 location="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties-1.2-draft-01.wsdl" />
```

```
<wsdl:import
    namespace=
    "http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-
1.2-draft-01.wsdl"

 location="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceLifetime-1.2-draft-01.wsdl" />


<wsdl:import
    namespace=

 "http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-
draft-01.wsdl"

 location="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-
BaseNotification-1.2-draft-01.wsdl" />


<!--============================================================

                     T Y P E S

  ============================================================-->
<types>
<xsd:schema
 targetNamespace="http://www.globus.org/namespaces/examples/core/
MathService_instance_rp"

 xmlns:tns="http://www.globus.org/namespaces/examples/core/
MathService_instance_rp"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"

 xmlns:wsrl="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceLifetime-1.2-draft-01.xsd"

 xmlns:wsbn="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-
BaseNotification-1.2-draft-01.xsd"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

        <!-- IMPORT WS-ADDRESSING -->
        <xsd:import
 namespace="http://schemas.xmlsoap.org/ws/2004/03/addressing"

 schemaLocation="http://schemas.xmlsoap.org/ws/2004/03/addressing"/>

        <!-- IMPORT WS-ResourceLifetime schemas -->
        <xsd:import
 namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceLifetime-1.2-draft-01.xsd"
```

```
 schemaLocation="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceLifetime-1.2-draft-01.xsd"/>

        <!-- IMPORT WS-BaseNotification schemas -->
        <xsd:import namespace="http://docs.oasis-open.org/wsn/2004/06/
wsn-WS-BaseNotification-1.2-draft-01.xsd"

 schemaLocation="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-
BaseNotification-1.2-draft-01.xsd"/>


        <!-- REQUESTS AND RESPONSES -->

        <xsd:element name="add" type="xsd:int"/>
        <xsd:element name="addResponse">
                <xsd:complexType/>
        </xsd:element>
        <xsd:element name="subtract" type="xsd:int"/>
        <xsd:element name="subtractResponse">
                <xsd:complexType/>
        </xsd:element>

        <xsd:element name="create">
                <xsd:complexType/>
        </xsd:element>
        <xsd:element name="createResponse">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element ref="wsa:EndpointReference"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>

        <!-- MathService Resource Properties -->

        <xsd:element name="Value" type="xsd:int"/>
        <xsd:element name="LastOp" type="xsd:string"/>
        <xsd:element name="MathResourceProperties">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element ref="tns:Value"/>
                    <xsd:element ref="tns:LastOp"/>

                    <xsd:element maxOccurs="1" minOccurs="1"
 ref="wsrl:CurrentTime"/>
                    <xsd:element maxOccurs="1" minOccurs="1"
 ref="wsrl:TerminationTime"/>

                    <!-- NotificationProducerRP -->
                    <xsd:element maxOccurs="unbounded" minOccurs="1"
 ref="wsbn:Topic"/>
                    <xsd:element maxOccurs="1" minOccurs="1"
 ref="wsbn:FixedTopicSet"/>
```

```
                    <xsd:element maxOccurs="unbounded" minOccurs="1"
 ref="wsbn:TopicExpressionDialects"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>

</xsd:schema>
</types>

<!--================================================================

                        M E S S A G E S

  ================================================================-->
<message name="AddInputMessage">
        <part name="parameters" element="tns:add"/>
</message>
<message name="AddOutputMessage">
        <part name="parameters" element="tns:addResponse"/>
</message>

<message name="SubtractInputMessage">
        <part name="parameters" element="tns:subtract"/>
</message>
<message name="SubtractOutputMessage">
        <part name="parameters" element="tns:subtractResponse"/>
</message>

<message name="CreateInputMessage">
        <part name="parameters" element="tns:create"/>
</message>
<message name="CreateOutputMessage">
        <part name="parameters" element="tns:createResponse"/>
</message>


<!--================================================================

                        P O R T T Y P E

  ================================================================-->

<portType name="MathPortType"
    wsrp:ResourceProperties="tns:MathResourceProperties">

 <operation name="add">
  <input message="tns:AddInputMessage"/>
  <output message="tns:AddOutputMessage"/>
 </operation>

 <operation name="subtract">
  <input message="tns:SubtractInputMessage"/>
  <output message="tns:SubtractOutputMessage"/>
 </operation>
```

```
 <operation name="createResource">
  <input message="tns:CreateInputMessage"/>
  <output message="tns:CreateOutputMessage"/>
 </operation>

        <!-- WS-ResourceProperties operations -->
 <operation name="GetResourceProperty">
  <input name="GetResourcePropertyRequest"
                    message="wsrpw:GetResourcePropertyRequest"

 wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties/GetResourceProperty"/>
                <output name="GetResourcePropertyResponse"
                    message="wsrpw:GetResourcePropertyResponse"

 wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties/GetResourcePropertyResponse"/>
                <fault name="InvalidResourcePropertyQNameFault"
                    message="wsrpw:InvalidResourcePropertyQNameFault"/
>
                <fault name="ResourceUnknownFault"
                    message="wsrpw:ResourceUnknownFault"/>
 </operation>

    <operation name="GetMultipleResourceProperties">
      <input name="GetMultipleResourcePropertiesRequest"
 message="wsrpw:GetMultipleResourcePropertiesRequest"
 wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties/GetMultipleResourceProperties"/>
      <output name="GetMultipleResourcePropertiesResponse"
 message="wsrpw:GetMultipleResourcePropertiesResponse"
 wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties/GetMultipleResourcePropertiesResponse"/>
      <fault name="InvalidResourcePropertyQNameFault"
 message="wsrpw:InvalidResourcePropertyQNameFault"/>
      <fault name="ResourceUnknownFault"
 message="wsrpw:ResourceUnknownFault"/>
    </operation>

    <operation name="SetResourceProperties">
      <input name="SetResourcePropertiesRequest"
 message="wsrpw:SetResourcePropertiesRequest" wsa:Action="http://
docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties/
SetResourceProperties"/>
      <output name="SetResourcePropertiesResponse"
 message="wsrpw:SetResourcePropertiesResponse" wsa:Action="http://
docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties/
SetResourcePropertiesResponse"/>
      <fault name="InvalidResourcePropertyQNameFault"
 message="wsrpw:InvalidResourcePropertyQNameFault"/>
      <fault name="UnableToModifyResourcePropertyFault"
 message="wsrpw:UnableToModifyResourcePropertyFault"/>
```

```
      <fault name="SetResourcePropertyRequestFailedFault"
 message="wsrpw:SetResourcePropertyRequestFailedFault"/>
      <fault name="ResourceUnknownFault"
 message="wsrpw:ResourceUnknownFault"/>
      <fault name="InvalidSetResourcePropertiesRequestContentFault"
 message="wsrpw:InvalidSetResourcePropertiesRequestContentFault"/>
    </operation>

    <operation name="QueryResourceProperties">
      <input name="QueryResourcePropertiesRequest"
 message="wsrpw:QueryResourcePropertiesRequest" wsa:Action="http://
docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties/
QueryResourceProperties"/>
      <output name="QueryResourcePropertiesResponse"
 message="wsrpw:QueryResourcePropertiesResponse" wsa:Action="http://
docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties/
QueryResourcePropertiesResponse"/>
      <fault name="UnknownQueryExpressionDialectFault"
 message="wsrpw:UnknownQueryExpressionDialectFault"/>
      <fault name="InvalidResourcePropertyQNameFault"
 message="wsrpw:InvalidResourcePropertyQNameFault"/>
      <fault name="QueryEvaluationErrorFault"
 message="wsrpw:QueryEvaluationErrorFault"/>
      <fault name="InvalidQueryExpressionFault"
 message="wsrpw:InvalidQueryExpressionFault"/>
      <fault name="ResourceUnknownFault"
 message="wsrpw:ResourceUnknownFault"/>
    </operation>

    <!-- WS-ResourceLifetime operations -->
    <!--     ScheduledResourceTermination -->
        <wsdl:operation name="SetTerminationTime">

            <wsdl:input message="wsrlw:SetTerminationTimeRequest"

 wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceLifetime/SetTerminationTime"/>
            <wsdl:output message="wsrlw:SetTerminationTimeResponse"

 wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceLifetime/SetTerminationTimeResponse"/>
            <wsdl:fault
                message="wsrlw:UnableToSetTerminationTimeFault"
                name="UnableToSetTerminationTimeFault"/>
            <wsdl:fault message="wsrlw:ResourceUnknownFault"
 name="ResourceUnknownFault"/>
            <wsdl:fault
                message="wsrlw:TerminationTimeChangeRejectedFault"
                name="TerminationTimeChangeRejectedFault"/>
        </wsdl:operation>

    <!--   ImmediateResourceTermination -->
        <wsdl:operation name="Destroy">
```

```
            <wsdl:input message="wsrlw:DestroyRequest"

 wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceLifetime/Destroy"/>
            <wsdl:output message="wsrlw:DestroyResponse"

 wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceLifetime/DestroyResponse"/>
            <wsdl:fault
                message="wsrlw:ResourceNotDestroyedFault"
                name="ResourceNotDestroyedFault"/>
            <wsdl:fault message="wsrlw:ResourceUnknownFault"
 name="ResourceUnknownFault"/>
        </wsdl:operation>


    <!-- NotificationProducerPort -->
        <wsdl:operation name="Subscribe">
          <wsdl:input message="wsbnw:SubscribeRequest"
 wsa:Action="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-
BaseNotification/Subscribe"/>
          <wsdl:output message="wsbnw:SubscribeResponse"
 wsa:Action="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-
BaseNotification/SubscribeResponse"/>
          <wsdl:fault name="TopicPathDialectUnknownFault"
 message="wsbnw:TopicPathDialectUnknownFault"/>
          <wsdl:fault name="SubscribeCreationFailedFault"
 message="wsbnw:SubscribeCreationFailedFault"/>
          <wsdl:fault name="ResourceUnknownFault"
 message="wsbnw:ResourceUnknownFault"/>
        </wsdl:operation>

</portType>

<!--============================================================

                      B I N D I N G

  ============================================================-->
<binding name="MathBinding" type="MathPortType">
    <soap:binding style="document"
 transport="http://schemas.xmlsoap.org/soap/http"/>

    <wsdl:operation name="add">
      <soap:operation soapAction=""/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>

    <wsdl:operation name="subtract">
```

```
  <soap:operation soapAction=""/>
  <wsdl:input>
    <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal"/>
  </wsdl:output>
</wsdl:operation>

<wsdl:operation name="createResource">
  <soap:operation soapAction=""/>
  <wsdl:input>
    <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal"/>
  </wsdl:output>
</wsdl:operation>

<wsdl:operation name="GetResourceProperty">
  <soap:operation soapAction=""/>
  <wsdl:input>
    <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal"/>
  </wsdl:output>
</wsdl:operation>

<wsdl:operation name="GetMultipleResourceProperties">
  <soap:operation soapAction=""/>
  <wsdl:input>
    <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal"/>
  </wsdl:output>
</wsdl:operation>

<wsdl:operation name="SetResourceProperties">
  <soap:operation soapAction=""/>
  <wsdl:input>
    <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal"/>
  </wsdl:output>
</wsdl:operation>

<wsdl:operation name="QueryResourceProperties">
  <soap:operation soapAction=""/>
  <wsdl:input>
    <soap:body use="literal"/>
  </wsdl:input>
```

```
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>

    <wsdl:operation name="SetTerminationTime">

      <soap:operation soapAction=""/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>

    <wsdl:operation name="Destroy">
      <soap:operation soapAction=""/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>

    <wsdl:operation name="Subscribe">
      <soap:operation soapAction=""/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>

</binding>

<!--============================================================

                      S E R V I C E

  ============================================================-->
<service name="MathService">
    <port name="MathPort" binding="MathBinding">
       <soap:address location="http://localhost:8080/wsrf/services/"/>
    </port>
</service>

</definitions>
```